МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ
И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

БАКАЛАВРСКАЯ РАБОТА

на тему: Разработка программного комплекса для анализа временных рядов с использованием искусственных нейронных сетей

Студент С.Д. Назаров _____

Руководитель, Н.И. Лиманова _____
д.т.н., профессор

Допустить к защите
Заведующий кафедрой к.тех.н, доцент, А.В. Очеповский _____

«_____»_____20_____г.

Тольятти 2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав.кафедрой «Прикладная
математика и информатика»
_____А.В.Очеповский

«____»_____2016 г.

ЗАДАНИЕ
на выполнение бакалаврской работы

Студент: Назаров Сергей Денисович

1. Тема: Разработка программного комплекса для анализа временных рядов с использованием искусственных нейронных сетей.

2. Срок сдачи студентом законченной выпускной квалификационной работы 19.06.2016 г.

3. Исходные данные к выпускной квалификационной работе: статически типизированный объектно-ориентированный язык программирования общего назначения C#, библиотека компонентов Encog, основанные на реальных данных обучающие и контрольные выборки.

4. Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов, разделов):

Введение

Глава 1. Теоретическое исследование нейросетевого анализа временных рядов

- 1.1 Основные понятия анализа временных рядов
- 1.2 Структурная модель прогнозирования на основе искусственных нейронных сетей
- 1.3 Анализ существующих разработок в сфере нейросетевого прогнозирования

Глава 2. Проектирование программного комплекса

2.1 Постановка задачи на разработку программного комплекса

2.2 Обоснование выбора средств разработки

2.3 Представление входных данных

2.4 Описание классовой структуры программного комплекса

2.5 Программная реализация искусственной нейронной сети

2.6 Программная реализация выделения трендовой компоненты временного ряда

2.7 Программная реализация расчёта ошибки прогнозирования

2.8 Описание графического интерфейса

Глава 3. Обучение и тестирование программного комплекса

3.1 Тестирование модуля «Обучение» и отбор оптимальных моделей искусственных нейронных сетей

3.2 Тестирование модуля «Прогноз»

3.3 Тестирование модуля «Тренд»

Заключение

Библиографический список

Приложения

5. Ориентировочный перечень графического и иллюстративного материала: диаграмма классов программного комплекса; графические представления искусственной нейронной сети; набор формул, объясняющих математический аппарат; демонстрация работы алгоритма на реальном наборе данных; графики и диаграммы, поясняющие результат работы системы.

6. Дата выдачи задания «11» января 2016 г.

Руководитель выпускной квалификационной работы _____ Н.И. Лиманова

Задание принял к исполнению _____ С.Д. Назаров

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

УТВЕРЖДАЮ
Зав. кафедрой «Прикладная
математика и информатика»
_____ А.В. Очеповский

«____»_____ 2016 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения бакалаврской работы

Студента Назарова Сергея Денисовича
по теме: Разработка программного комплекса для анализа временных рядов с использованием искусственных нейронных сетей

Наименование раздела работы	Плановый срок выполнения раздела	Фактический срок выполнения раздела	Отметка о выполнении	Подпись руководителя
Поиск и исследование литературы по теме бакалаврской работы	21.02.2016	21.02.2016	выполнено	
Написание первой главы бакалаврской работы	25.02.2016	25.02.2016	выполнено	
Написание второй главы бакалаврской работы	11.03.2016	11.03.2016	выполнено	
Разработка нейросетевого алгоритма для анализа временного ряда	08.04.2016	08.04.2016	выполнено	
Тестирование и отладка приложения	16.04.2016	16.04.2016	выполнено	
Написание третьей главы бакалаврской работы	19.04.2016	19.04.2016	выполнено	
Оформление текста бакалаврской работы	15.05.2016	15.05.2016	выполнено	

Представление работы на кафедру	16.05.2016	16.05.2016	выполнено	
Подготовка доклада и графического материала	22.05.2016	22.05.2016	выполнено	
Предварительная защита бакалаврской работы	25.05.2016	25.05.2016	выполнено	
Сдача пояснительной записки ВКР	19.06.2016	19.06.2016	выполнено	

Руководитель выпускной квалификационной работы _____ Н.И. Лиманова

Задание принял к исполнению _____ С.Д. Назаров

Аннотация

Тема: Разработка программного комплекса для анализа временных рядов с использованием искусственных нейронных сетей.

Актуальность темы данной работы заключается в поиске методов более точного и быстрого анализа данных и их прогнозирования.

Целью данной бакалаврской работы является разработка программного комплекса для анализа временных рядов на основе искусственных нейронных сетей.

Для достижения поставленной цели в ходе работы были поставлены и решены следующие задачи:

– проанализировать существующие решения в данной области;

– проанализировать технологии для реализации программного комплекса;

– разработать модель искусственной нейронной сети;

– реализовать программный комплекс в виде оконного приложения;

– провести тестирование.

Работа состоит из введения, трех глав, заключения, списка использованной литературы и приложений.

Первая глава посвящена анализу предметной области и обзору существующих решений.

Вторая глава посвящена обоснованию выбора средств для разработки, проектированию и разработке программного комплекса.

Третья глава посвящена тестированию программного комплекса.

В заключении сформулированы основные выводы, которые были сделаны в процессе написания бакалаврской работы, описаны результаты практической реализации проекта.

Выпускная квалификационная работа содержит пояснительную записку объёмом 59 страниц, включая 22 рисунка, две таблицы и четыре приложения.

Оглавление

Введение

В настоящее время для изучения свойств сложных систем, в том числе и при экспериментальных исследованиях, широко используется подход, основанный на анализе сигналов, произведенных системой. Это очень актуально в тех случаях, когда математически описать изучаемый процесс практически невозможно, но в нашем распоряжении имеется некоторая характерная наблюдаемая величина. В данной работе описывается интервальное прогнозирование таких процессов и выделение их основных характеристик на базе набора статистических данных. В качестве модели представления этих данных взят временной ряд, который принято моделировать, используя понятие случайного процесса

Под случайным (стохастическим) процессом понимают семейство случайных величин $\{\xi_t\}_{t\in T}$, $T \in R$, где параметр интерпретируется как время [9].

Временной ряд – это одна из возможных реализаций случайного процесса на некотором ограниченном промежутке индексируемого параметра (времени) [2].

Временные ряды являются очень удобным механизмом описания процессов, работающих по принципу «черный ящик». В таких системах мы не имеем возможности «заглянуть» внутрь процесса, а видим лишь конечный результат. Другими словами, временные ряды позволяют анализировать данные, предметная область которых плохо изучена или неизвестна.

Можно выделить две основные цели анализа временных рядов: определение природы ряда (выделение компонентов, оценка их параметров) и использование полученных оценок для целей прогнозирования. Для определения характеристик ряда используются экспертные и математические методы, но для прогнозирования будущих значения ряда подобные решения не

будут столь эффективными. Группа экспертов может не принять во внимание все факторы, а использование математических формул при анализе больших массивов данных требует таких же больших вычислительных мощностей и часто не является оптимальным средством для прогнозирования в деятельности предприятий.

Технологии искусственного интеллекта и, в частности, системы базирующиеся на искусственных нейронных сетях в последние годы все активнее используются для прогнозирования временных рядов. Отличие этого подхода от стандартных состоит в том, что он позволяет сделать систему самообучаемой, что особенно важно для трудноформализуемых задач. Благодаря возможности работы с «зашумленными» данными система получается гибкой, хотя обычно не решает задачу со 100%-ной точностью.

Предметной областью, в которой уже получили признание нейросетевые алгоритмы, является сфера экономики и финансов, где любая помощь в принятии решений может оптимизировать трудовые и материальные ресурсы. В этой области нейросетевые алгоритмы нашли своё применение в форме математического ядра интеллектуальных систем принятия решений, экспертных систем, оболочек для имитационного моделирования, нейросетевых баз знаний и др. В данной бакалаврской работе внимание акцентируется на одном из направлений использования нейросетей – это применение нейросетевых технологий для прогнозирования значений временных рядов.

Задача прогнозирования временных рядов имеет высокую актуальность для многих предметных областей и является неотъемлемой частью деятельности многих компаний. Любая помощь в принятии решений в бизнес-процессах, может помочь уменьшить затраты, оптимизировать трудовые и материальные ресурсы, а искусственные нейронные сети – мощный инструмент в задачах подобного рода. Самообучающиеся алгоритмы – это самое перспективное направление в области анализа данных, а задача

прогнозирования временных рядов – самая популярная задача машинного обучения.

Для того, чтобы применить технологии искусственного интеллекта для решения практических задач связанных с анализом и прогнозированием временного ряда, необходимо решить следующие задачи:

1. Провести анализ предметной области в области анализа временных рядов.
2. Проанализировать сущность комплекса задач по прогнозированию временных рядов и выделения их компонент.
3. Обосновать использование программных средств.
4. Построить нейросетевую модель.
5. Охарактеризовать входные, постоянные, промежуточные и выходные данные.
6. Разработать графический пользовательский интерфейс.
7. Реализовать выбранный вариант проекта.
8. Провести обучение и тестирование искусственной нейронной сети.

Объектом исследования являются нейросетевая модель прогнозирования будущих значений временного ряда.

Предметом исследования является поддержка принятия решений при анализе данных путём создания программного комплекса для анализа временных рядов.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложений.

Во введении описывается актуальность проводимого исследования, формируется цель и ставятся задачи, которые необходимо решить для достижения цели.

В первой главе описываются теоретические аспекты анализа временных рядов, рассматриваются структурные компоненты ряда и даётся определение понятию прогнозирование. Изучаются основные сведения о искусственных

нейронных сетях и их обучении, рассматриваются существующие программные решения.

Во второй главе проводится разработка и реализация программного комплекса для анализа временных рядов, ставятся задачи на разработку и обосновывается выбор средств разработки. Моделируется и реализуется код искусственной нейронной сети и описывается интерфейс программного комплекса.

В третьей главе проводится тестирование созданного программного комплекса с разными исходными параметрами. Проверяется эффективность обучения, проводится сравнительный анализ точности прогнозирования для разных моделей искусственных нейронных сетей. Тестируется выделение трендовой компоненты и сравнивается точность прогнозирования при различной длительности обучения.

В заключении приводятся основные выводы по работе, достигнутые в ходе выполнения выпускной квалификационной работы.

Глава 1 Теоретическое исследование нейросетевого анализа временных рядов

1.1 Основные понятия анализа временных рядов

Понятие анализ временных рядов используется для того, чтобы отделить эту задачу от более простых задач анализа данных (когда нет естественного порядка поступления наблюдений) и, во-вторых, от анализа пространственных данных, в котором наблюдения зачастую связаны с географическим положением. Временные ряды могут быть представлены как графически, так и в табличном виде.

Временные ряды отличаются от простых статистических выборок в фиксированный момент времени следующими признаками:

– последовательные во времени показатели временных рядов являются взаимозависимыми, особенно это относится к близко расположенным наблюдениям;

– в зависимости от момента наблюдения показатели временного ряда обладают разной информативностью: информационная ценность наблюдений убывает по мере их удаления от текущего момента времени;

– с увеличением количества показателей временного ряда точность статистических характеристик не будет увеличиваться пропорционально числу наблюдений, а при появлении новых закономерностей развития она может даже уменьшаться.

Основное правило построения временных рядов – необходимость обеспечения сопоставимости его отдельных показателей. Для этого все элементы должны характеризовать изучаемое явление за равные промежутки времени или фиксировать состояние признака через равные интервалы. Каждое значение показателя во временном ряду необходимо рассчитывать по единой методике и выражать в одних и тех же единицах измерения. Количество

измерений должно быть достаточно представительным, чтобы выявить устойчивую тенденцию. В то же время следует учитывать, что использование слишком большого количества ретроспективных значений может привести к преувеличению прошлых тенденций, нечувствительности тренда к переменам.

1.1.1 Методы определения структуры временного ряда

В процессе принятия решений, эффективность решений часто зависит от количества доступной информации об изучаемой системе. Для более эффективного анализа и прогнозирования временных рядов, необходимо изучить исходные графики и выделить их основные составляющие. Определение структуры и составляющих временного ряда может быть использовано как своеобразная «разведка» данных. Например, знание о наличии сезонной компоненты необходимо для определения количества записей выборки, которое должно принимать участие в построении прогноза.

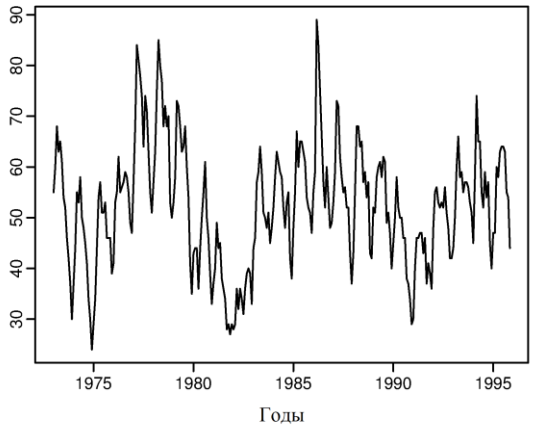
Определим компоненты, которые принято выделять во временных рядах [14]:

1. Тренд – динамика, характеризующее общее развитие временного ряда.
2. Циклическая компонента – динамика, имеющая фазу возрастания и убывания, период которой занимает достаточно большой промежуток времени.
3. Сезонная компонента – это регулярные колебания уровней ряда в определенное время суток, недели, сезона и т.д.
4. Календарные эффекты – скачки временного ряда, связанные с некоторыми предсказуемыми календарными событиями (например, праздниками или выходными)
5. Аномальные явления (выбросы) – непредсказуемые скачки, приводящие к резким, но кратковременным отклонениям ряда от общей тенденции развития.

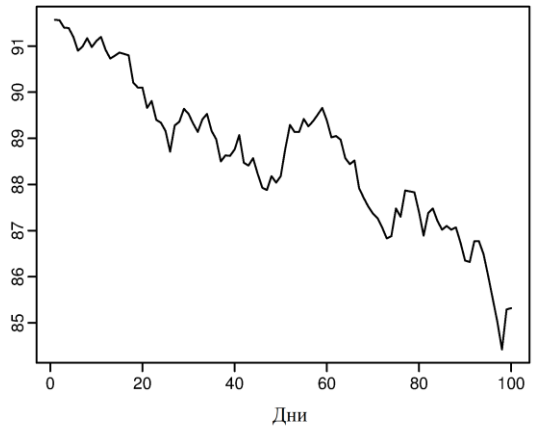
6. Структурные сдвиги – непредсказуемые скачки, приводящие к отклонениям ряда от общей тенденции развития, которые сказываются на всем его дальнейшем поведении.

7. Случайная компонента – беспорядочные движения достаточно большой частоты, связанные с влиянием большого количества неизвестных факторов.

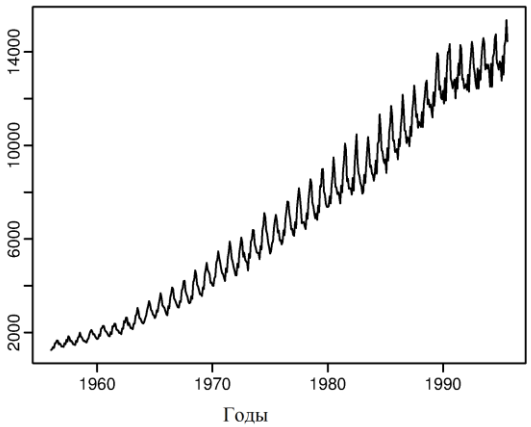
Некоторые временные ряды представляют собой ту или иную компоненту в чистом виде, однако на практике такие ряды встречаются крайне редко. В общем, временной ряд может представлять собой аддитивную, мультипликативную или смешанную комбинацию некоторых из этих компонент. Примеры смешивания компонент приведены на рисунке 1.1 [16]. Кроме того, далеко не все временные ряды имеют достаточно простую структуру для разложения на указанные компоненты.



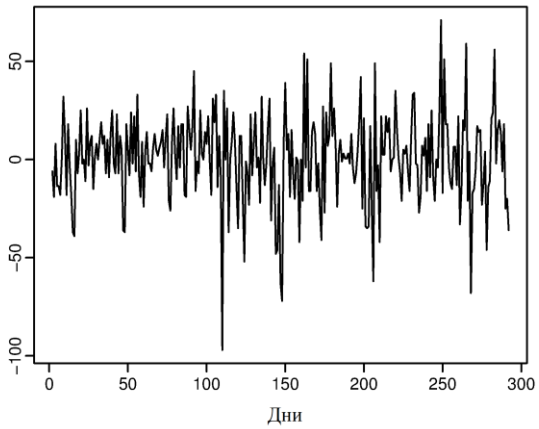
1. Продажа жилья (в миллионах)



2. Спрос на товар



3. Производство электроэнергии



4. Изменение индекса Доу Джонса

Рисунок 1.1 – Примеры временных рядов основанных на реальных данных. 1) сезонная компонента и циклическая компонента каждые 6-10 лет; 2) убывающий тренд; 3) возрастающий тренд и сезонная компонента; 4) случайная компонента

Анализ временного ряда производится на основе построенной модели, параметры которой подлежат оценке. Большинство регулярных составляющих временных рядов принадлежит к двум классам: они являются либо трендом, либо сезонной составляющей.

Тренд является систематической компонентой временного ряда, которая может изменяться во времени. Трендом называют неслучайную функцию, которая формируется под действием общих или долговременных тенденций, влияющих на временной ряд [15]. Примером тенденции может выступать, например, фактор роста исследуемого рынка.

При анализе тренда выделают четыре его основных вида:

1. Полиномиальный тренд. Применяют для описания временных рядов с плавной, медленной изменяющейся со временем динамикой.

2. Экспоненциальный тренд. Целесообразно применять для быстро растущих временных рядов.

3. Гармонический тренд. Применение гармонического тренда оправдано, если в поведении временного ряда отчётливо просматривается периодичность.

4. Тренд, выраженный логистической функцией. Выгодно отличается от других тем, что имеет асимптоту. Часто встречается при анализе временных рядов демографических показателей.

Функции четырёх основных видов трендов указаны в таблице 1.1, где A – амплитуда колебаний, f – частота, φ – сдвиг по фазе, k – асимптота.

Таблица 1.1 – Основные виды трендов и их функции

Полиномиальный	$y_t = a_0 + a_1 t + \cdots a_p t^p$
Экспоненциальный	$y_t = e^{a_0 + a_1 t + \cdots a_p t^p}$

Гармонический	$y_t = A\cos\left(2\pi ft + \varphi\right)$
Логистический	$y_t = \frac{k}{1 + be^{-at}}$

Для оценки параметров тенденций первого и второго вида достаточно применить метод наименьших квадратов (МНК), который по сути сводится к решению системы линейных уравнений.

Параметры тренда третьего типа в общем случае не могут быть оценены с помощью МНК, однако, если частота известна, гармонический тренд несложно представить в виде линейной комбинации sin и cos указанной в формуле 1.1.

$$A\cos(2\pi ft + \varphi) = \alpha\cos(2\pi ft) + \beta\cos(2\pi ft) \quad (1.1)$$

Далее можно применить МНК для оценки параметров α и β .

Задача МНК заключается в нахождении коэффициентов линейной зависимости, при которых функция двух переменных α и β формула $F(\alpha, \beta) = \sum_{i=1}^{n}(y_i - (\alpha x_i + b))^2$ принимает наименьшее значение [8]. То есть, при данных a и b сумма квадратов отклонений экспериментальных данных от найденной прямой будет наименьшей. Таким образом, решение сводится к нахождению экстремума функции двух переменных.

1.1.2 Постановка задачи прогнозирования будущих значений временного ряда

Прогнозирование – это предсказание будущих событий. Целью прогнозирования является уменьшение риска при принятии решений. Рассмотрим следующую задачу: существует система, работающая по принципу «чёрный ящик». Система, которую представляют как «черный ящик», рассматривается как имеющая некий «вход» для ввода информации и «выход» для отображения результатов работы, при этом происходящие в ходе работы системы процессы наблюдателю неизвестны. Предполагается, что состояние выходов функционально зависит от состояния входов.

На вход такой системе подаётся набор сигналов $(x_1, \dots, x_m)$, а на выходе мы имеем некоторую величину $y(x_1, \dots, x_m)$. Результат работы такой системы можно описать с помощью матрицы, описанной в формуле 1.4.

$$(Y, X) = \begin{pmatrix} y_1 & x_{11} & \cdots & x_{1m} \\ y_2 & x_{21} & \cdots & x_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ y_n & x_{m1} & \cdots & x_{mm} \end{pmatrix} \quad (1.4)$$

Здесь $(x_{i1}, \dots, x_{im})$ – набор управляющих сигналов, иначе называемых регрессорами, в i -ом измерении, y_i – результат i -ого измерения. Для того чтобы построить прогноз по этим данным, необходимо выявить зависимость между набором управляющих сигналов и значениями на выходе, то есть найти некоторую функцию g из формулы 1.5.

(1.5)

Теоретическим обоснованием такого подхода является теорема Такенса. Данная теорема гласит, что если временно ряд порождается динамической системой, т.е. множество значений временного ряда есть произвольная функция состояния такой системы, то существует такое число *d* (примерно равно эффективному числу степеней свободы данной динамической системы), что *d* предыдущих значений временного ряда однозначно определяют следующее значение.

На практике большинство прогнозируемых временных рядов порождаются сложными динамическими системами, с множеством влияющих на них факторов, поэтому *d* для них велико.

В настоящее время существует более 100 моделей прогнозирования временных рядов, но среди них встречается большое количество вариаций нескольких базовых. Подобные базовые модели можно разбить на две большие группы: статические и структурные.

В статических моделях зависимость будущих значений от прошлых задаётся уравнениями. К ним относятся: регрессионные модели, авторегрессионные модели, модели экспоненциального сглаживания, модели по выборке максимального подобия и т.д.

Выбор модели должен основываться на типе используемых данных. Например, регрессионная модель использует значения большого количества факторов, влияющих на конечное значение случайного процесса. Чем больше таких факторов, тем точнее прогноз. Авторегрессионные модели предполагают, что значение процесса зависит от некоторых предыдущих значений того же процесса, а экспоненциальное сглаживание постоянно пересматривает прогнозные значение по мере поступления фактических.

Поскольку некоторые зависимости могут иметь очень сложный вид, на практике такую функцию удаётся найти далеко не всегда. Поэтому ограничиваются поиском некоторой аппроксимирующей зависимости. Именно это и делают структурные модели прогнозирования, которые задают зависимость в виде структуры и правил перехода по ней. По такому принципу работают искусственные нейронные сети, модели на базе цепей Маркова, модели на базе классификационно-регрессионных деревьев и т.д.

1.2 Структурная модель прогнозирования на основе искусственных нейронных сетей

Искусственные нейронные сети (ИНС) – математические модели, а также их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [13]. Чтобы построить математическую модель процессов, происходящих в мозгу, примем несколько предположений:

– каждый нейрон обладает некоторой передаточной функцией, определяющей условия его возбуждения в зависимости от силы полученных сигналов; кроме того, передаточные функции не зависят от времени;

– при прохождении синапса сигнал меняется линейно, т.е. сила сигнала умножается на некоторое число; это число называют «весом» синапса или весом соответствующего входа нейрона;

– деятельность нейронов синхронизирована, т.е. время прохождения сигнала от нейрона к нейрону фиксировано и одинаково для всех связей; то же самое относится ко времени обработки принятых сигналов.

Необходимо заметить, что веса синапсов могут меняться со временем – это принципиальная особенность, которая даёт неоспоримое преимущество этой модели прогнозирования над другими.

ИНС способны извлекать скрытые закономерности из потока данных. При этом данные могут быть неполны, противоречивы и даже заведомо искажены. Если между входными и выходными данными существует какая-то связь, пусть даже не обнаруживаемая традиционными корреляционными методами, ИНС способна автоматически настроиться на нее с заданной степенью точности [4].

ИНС можно рассматривать как сеть «нейронов», организованных в слоях. Предикторы (или входные нейроны) образуют нижний слой, а прогнозы (или выходы) формируют верхний слой. Могут быть и промежуточные слои, содержащие «скрытые нейроны».

Самые простые сети не содержат скрытых слоев и эквивалентны линейной регрессии. В этом случае регрессия будет более эффективным методом для обучения модели. После добавления промежуточного слоя со скрытыми нейронами, ИНС становится нелинейной. Нелинейная ИНС демонстрируется на рисунке 1.2.

Это известно, как многослойная однонаправленная сеть, где каждый слой узлов принимает входные сигналы от предыдущих слоев. Выходы узлов в одном слое являются входами к следующему слою. Входы каждого узла объединяются с использованием взвешенной линейной комбинации. Затем результаты модифицируются нелинейной функцией перед выводом. Количество скрытых слоев и количество узлов в каждом скрытом слое, должны быть точно определены заранее.

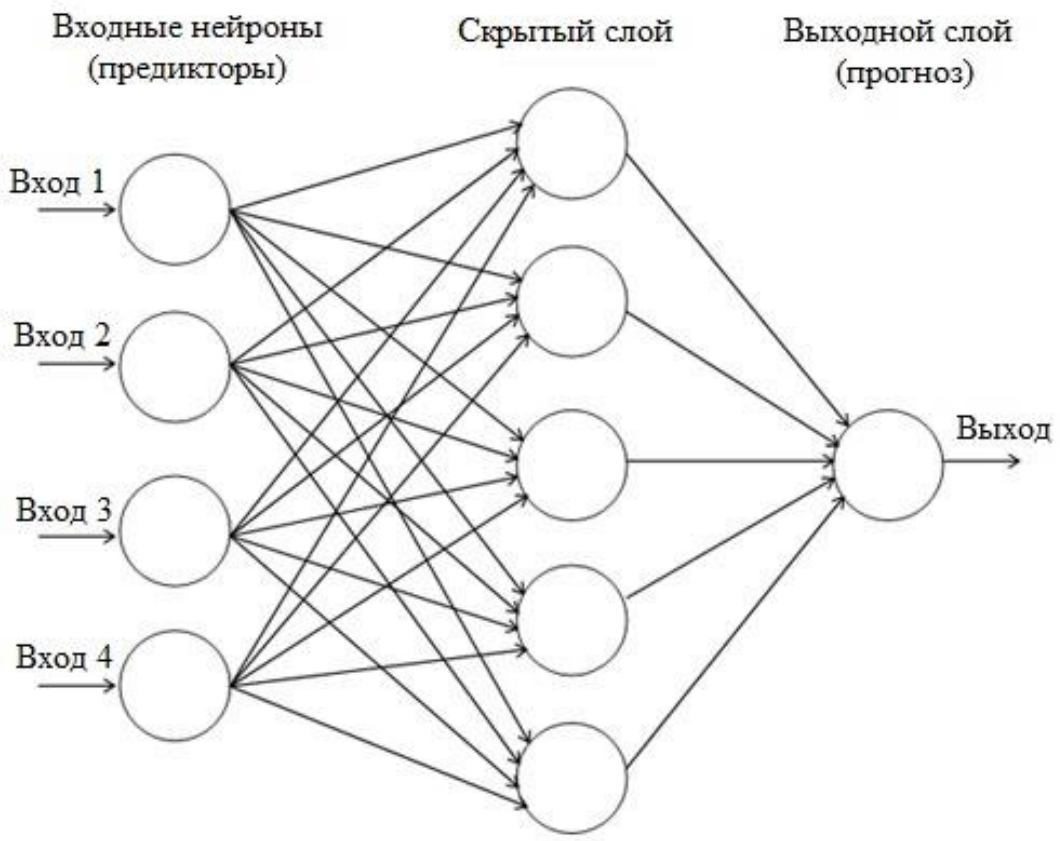


Рисунок 1.2 – Модель нелинейной ИНС

1.2.1 Методы обучения искусственных нейронных сетей

В начале работы ИНС, её веса принимают случайные значения и обновляются с использованием наблюдаемых данных. Следовательно, в прогнозах, полученных с помощью ИНС присутствует элемент случайности. Как правило, работу ИНС проверяют несколько раз с использованием различных входных данных и результаты усредняются. Процесс усреднения весов искусственных нейронов с целью получить желаемый результат на выходе, называется обучением ИНС, а входные данные для обучения – обучающими примерами или обучающей выборкой.

Обучающая выборка – выборка, по которой строится модель зависимостей. Желаемый результат называется тестовой выборкой. По нему оценивается качество моделей зависимостей. Оценку качества производят для выбора наилучшей модели. Для эффективной проверки модели, её проверяют на данных, которые не участвовали в обучении.

Обычно формирование обучающих примеров осуществляется по принципу «скользящего окна»: т.е. берется некоторый отрезок временного ряда и из него выделяется несколько наблюдений, которые и будут представлять собой входной вектор. Значением желаемого выхода в обучающем примере будет следующее по порядку наблюдение. Затем «скользящее окно» сдвигается на одну позицию в направлении возрастания времени, и процесс формирования следующей пары обучающей выборки повторяется.

1.3 Анализ существующих разработок в сфере нейросетевого прогнозирования

1.3.1 Анализ существующих программных продуктов

Сегодня разработано большое количество программных продуктов, пригодных для применения там, где возникает необходимость использования технологии нейровычислений. Существуют универсальные нейросетевые пакеты, предназначенные для решения любых задач, которые можно решить при помощи ИНС, от распознавания речи и образов до решения задач прогнозирования, но, как показывает практика, такие программные продукты не всегда удобны для решения задач прогнозирования временных рядов. Существует класс нейросетевых программных продуктов, предназначенных исключительно для решения задач прогнозирования временных рядов. Наиболее популярные сегодня следующие программные продукты, реализующие нейросетевые подходы к решению задач прогнозирования.

1. Matlab – настольная лаборатория для математических вычислений, проектирования электрических схем и моделирования сложных систем. Имеет встроенный язык программирования и весьма богатый инструментарий для нейронных сетей – AnfisEditor (обучение, создание, тренировка и графический интерфейс), командный интерфейс для программного задания сетей, nnTool – для более тонкой конфигурации сети.

2. Statistica – мощнейшее обеспечение для анализа данных и поиска статистических закономерностей. В данном пакете работа с нейросетями

представлена в модуле STATISTICA Neural Networks (сокращенно, ST Neural Networks, нейронно-сетевой пакет фирмы StatSoft), представляющий собой реализацию всего набора нейросетевых методов анализа данных.

3. BrainMaker – предназначен для решения задач, для которых пока не найдены формальные методы и алгоритмы, а входные данные неполны, зашумлены и противоречивы. К таким задачам относятся биржевые и финансовые предсказания, моделирование кризисных ситуаций, распознавание образов и многие другие.

4. NeuroShell DayTrader – нейросетевая система, ориентирована для использования в трейдерской деятельности. Программа является узкоспециализированной и подходит для торговли, но по своей сути слишком близка к черному ящику.

Каждое из рассматриваемых решений является дорогостоящим коммерческим продуктом. Например, пакет прикладных программ Matlab оценивается в 2650$, а модуль Neural Network Toolbox в 1250$. Подобные программные продукты почти недоступны для личного пользования и затрудняют использование современных технологий искусственного интеллекта в бизнес-процессах коммерческих предприятий. Кроме того, многие решения на рынке не позволяют использовать исходный код смоделированной ИНС для собственных разработок.

1.3.2 Анализ существующих библиотек компонентов

Библиотека в программировании — сборник подпрограмм или объектов, используемых для разработки программного обеспечения [2]. Основным назначением представленных ниже библиотек является интеграция нейросетевых технологий в собственные информационные и программные системы, для расширения их аналитических возможностей. Реализация ИНС в виде компонентов и наличие открытого кода позволяет легко встраивать подобные библиотеки в другие программы. Объектно-ориентированное исполнение придает им особую гибкость для оптимизации под любые задачи.

Самые распространённые нейросетевые библиотеки для работы с временными рядами:

1. ANN – бесплатная библиотека с открытым исходным кодом для создания ИНС на языке PHP 5.x. Исходный код основан на работе Эдди Янга, в которой последние изменения вносились в 2002 году. С 2007 года проект развивает и поддерживает Томас Виен. ANN является свободно распространяемой библиотекой при условии сохранения информации об авторстве в исходном коде.

2. FANN – это бесплатная библиотека с открытым исходным кодом для создания ИНС. Сегодня библиотека FANN доступна почти для всех языков программирования и сред разработки. Она достаточно проста в использовании, универсальна и очень хорошо документирована (ссылку на сайт и на онлайн-документацию см. в архиве). В справочном руководстве подробно расписаны используемые классы, методы, свойства и т.д. с большим количеством обучающих примеров, но, это все доступно только на английском языке.

3. Encog – бесплатная библиотека для создания ИНС на языках Java, C# и C++. Encog поддерживает различные алгоритмы обучения, такие как байесовские сети, и скрытые Марковские модели. Тем не менее, его основная сила заключается в нейросетевых алгоритмах. Encog содержит классы для создания сетей с различными архитектурами, а также вспомогательные классы для нормализации и обработки данных ИНС.

4. NeuralBase – это библиотека с открытым исходным кодом, реализующая сеть Хопфилда и многослойную нейронную сеть обучаемую по алгоритму обратного распространения. Библиотека является интеллектуальной собственностью компании BaseGroup Labs – профессионального поставщика программных продуктов и решений в области анализа данных.

Разнообразие доступных библиотек позволяет использовать их в решении любой задачи независимо от выбранного языка программирования и архитектуры ИНС. Открытый исходный код даёт возможность для тонкой

настройки разрабатываемой программной системы, что особенно важно в задачах о принятии решений.

1.3.3 Обоснование необходимости разработки программного комплекса

Коммерческие программные продукты могут быть труднодоступны для пользователей и организация работающих с временными рядами, а работа с программным кодом и библиотеками компонентов может потребовать высокой квалификации сотрудника. В процессе принятие решений, пользователю может потребоваться как результат работы прогнозирующей системы, так и графическая демонстрация основных компонент временного ряда для возможности дополнительной поддержки решения с помощью экспертных систем. Рассмотренные выше программные продукты содержат лишь часть этих функций и достаточно сложны в освоении для конечного пользователя.

В процессе анализа существующих программных продуктов, было принято решение создать программный комплекс в видео оконного приложения, имеющий возможность прогнозировать, обучаться на данных любого типа и графически отображать рассматриваемый временной ряд. Программный комплекс должен иметь интуитивно понятный интерфейс. Средства для разработки программного комплекса будут избираться исходя из требования их надёжности и скорости их работы, для максимально быстрой и удобной работы с массивами данных.

Глава 2 Проектирование программного комплекса

2.1 Постановка задачи на разработку программного комплекса

Программный комплекс – это набор взаимодействующих программ:

- согласованных по функциям и форматам;
- имеющих единообразные, точно определенные интерфейсы;
- составляющих полное средство для решения больших задач [10].

Задача разрабатываемого программного комплекса – помощь в принятии решений при работе с данными имеющими возможность быть оформленными в виде интервального временного ряда.

Программный комплекс должен состоять из трёх модулей:

1. Модуль определяющий тренд временного ряда с помощью математических методов. Модуль должен графически представлять исходные данные и выделенный тренд.

2. Модуль обучения ИНС. Модуль должен предоставлять пользователю данные о каждой итерации в процессе обучения. Под данными понимаются: номер итерации, значение ошибки прогнозирования, алгоритм обучения.

3. Модуль прогнозирования будущих значений. Модуль должен предоставлять пользователю данные о каждом спрогнозированном значении. Под данными понимаются: дата прогнозируемого события, значение, полученное на выходе ИНС, реальное значение, значение ошибки прогнозирования.

Помимо этого, в программном комплексе должны быть реализованы следующие функции:

1. Возможность выбора файла с входными данными в файловой системе компьютера.

2. Возможность изменять количество скрытых слоёв ИНС и количество нейронов в каждом слое.

3. Возможность фильтрации входных значений («от» и «до» определённых дат) для модулей обучения и прогнозирования.

4. Сохранения и загрузки параметров обученной ИНС.

5. Экспорт результатов прогнозирования.

Выделение тренда временного ряда будет происходить с помощью математического метода наименьших квадратов.

Задача прогнозирования входных значений представляет собой, по существу, задачу прогнозирования временных рядов, которая в свою очередь сводится к задаче аппроксимации одномерных функций [3]. При этом можно использовать либо одно предыдущее значение целевой переменной x_{t-1}, либо несколько предыдущих: $x_{t-1}, x_{t-2}, \dots$. В этом случае увеличивается количество входов ИНС.

Общая схема нейросетевого предсказания временных рядов включает следующие этапы:

1. Определение временного интервала. Формирование базы данных.

2. Кодирование входов – выходов (ИНС могут работать только с числами).

3. Нормировка данных (результаты нейроанализа не должны зависеть от выбора единиц измерения).

4. Предобработка данных (удаление очевидных регулярностей из данных облегчает нейронным сетям выявление нетривиальных закономерностей).

5. Обучение нескольких ИНС с раз личной архитектурой (результат обучения зависит как от размеров сети, так и от ее начальной конфигурации).

6. Отбор оптимальных сетей – тех, которые дадут наименьшую ошибку предсказания.

7. Адаптивное предсказание и принятие решений.

Таким образом, перед началом моделирования ИНС и разработкой программного комплекса необходимо определить, сколько предыдущих значений одной переменной взять и как далеко вперед прогнозировать значение

выходной переменной. Наблюдения должны быть упорядочены во времени, поскольку их порядок имеет значение.

2.2 Обоснование выбора средств разработки

2.2.1 Обоснование выбора языка программирования

Для реализации программного комплекса для анализа временных рядов был изучен объектно-ориентированный язык программирования C#. Он относится к языкам C-подобным синтаксисом. Преимущества языка программирования C# над другими:

1. Подлинная объектная ориентированность (всякая языковая сущность претендует на то, чтобы быть объектом).

2. Компонентно-ориентированное программирование.

3. Безопасный код по сравнению с языками C и C++.

4. Унифицированная система типизации в которой все типы языка вплоть до примитивных наследуются от единого корневого объекта (от класса Object).

5. Поддержка событийно-ориентированного программирования.

6. «Родной» язык для создания приложений в среде .NET.

7. Объединение лучших идей современных языков программирования: Java, C++, Visual Basic и др.

К недостаткам можно отнести слабо развитую кроссплатформенность и относительно невысокую производительность относительно языков C и C++.

По существу, рекомендаций по выбору оптимального языка для программирования ИНС не существует. Код ИНС и алгоритмы обучения используют в своей основе простейшие структуры языка, поэтому для программирования можно использовать почти любой доступный язык, включая специализированные языки, встроенные в программные пакеты. Выбор языка программирования должен обуславливаться требуемой скоростью работы, наличием готовых решений и предполагаемым конечным видом программного решения.

Поскольку язык C# удовлетворяет всем функциональным условиям и хорошо себя показывает в тестах производительности [1], то именно он был выбран в качестве языка программирования для реализации программного комплекса.

2.2.2 Обоснование выбора среды разработки

Основными требованиями к выбору среды разработки является поддержка языка программирования C# и возможность проектирования графических интерфейсов для оконных приложений.

Рассмотрим популярные программные продукты, удовлетворяющие условиям:

1. SharpDevelop — это бесплатная среда программирования для проектов на платформе Microsoft .NET. SharpDevelop позволяет программировать на языках C#, VB.NET, Boo, IronPython, IronRuby, F#. Это IDE с открытым исходным кодом, можно свободно скачать исходный код и исполняемые файлы с сайта загрузки.

2. MonoDevelop является бесплатной кроссплатформенной IDE для языков C#, Visual Basic .NET и других языков .NET. В MonoDevelop можно быстро писать настольные приложения и ASP.NET веб-приложения для операционных систем Linux, Windows и Mac OSX. MonoDevelop делает легким для разработчиков портирование .NET приложений, созданных с помощью MS Visual Studio в Linux и Mac OSX, а также обеспечивает поддержку единого кода для всех платформ.

3. Microsoft Visual Studio позволяет создавать приложения Windows Forms, WPF, консольные приложения и библиотеки классов, а также проекты Win32, библиотеки классов и приложения среды CLR с использованием C#. Имеется возможность объединять проекты на разных языках в одно большое единое решение. Visual Studio насыщена функциональностью улучшения структуры кода, анализа и мощными средствами отладки во время выполнения приложений.

Решающим фактором при выборе стал интуитивно понятный инструмент для разработки пользовательского графического интерфейса Windows Forms интегрированный в среду разработки Visual Studio и изображённый на рисунке 2.1.

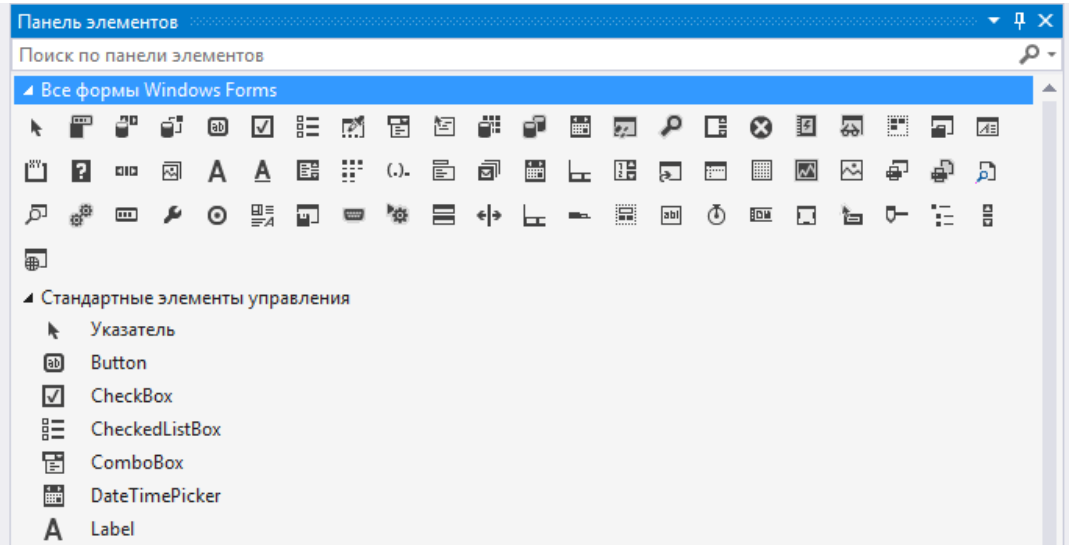


Рисунок 2.1 – Панель выбора элементов в Windows Forms

Благодаря наличию инструмента Windows Forms в качестве интегрированной среды разработки был выбран бесплатный программный продукт корпорации Microsoft – Visual Studio 2015 Community. Данная среда разработки представляет собой полный набор инструментов для создания как настольных приложений, так и корпоративных веб-приложений для совместной работы групп. Используя эффективные инструменты разработки Visual Studio, основанные на использовании компонентов, и другие технологии, можно не только эффективно работать с настольными приложениями, но и упрощать совместное проектирование, разработку и развертывание корпоративных решений [6].

2.2.3 Обоснование использования библиотек компонентов

Для программной реализации архитектуры ИНС и значений можно использовать готовые бесплатные библиотеки компонентов, предназначенных для работы с временными рядами. Для этих целей была выбрана библиотека

ENCOG – пакет для работы с нейросетями и системой машинного обучения, разработанной Heaton Research. Основные преимущества ENCOG перед аналогичными решениями:

1. Библиотека ENCOG является бесплатной, ее исходные коды доступны. При необходимости изучить структуру ИНС, можно посмотреть исходный код.

2. Пакет ENCOG хорошо документирован основателем компании Heaton Research. Существует доступный бесплатно онлайн-курс по нейронным сетям, машинному обучению и использовании ENCOG для предсказания будущих данных. В дополнение к этому, на сайте Heaton Research доступны книги по программированию с использованием ENCOG на языках Java и C#. На сайте разработчика также доступна онлайн-документация по ENCOG.

3. ENCOG имеет активную поддержку со стороны разработчика. На момент написания бакалаврской работы доступна версия ENCOG 3.3 и ведётся разработка версии 3.4 с поддержкой глубокого обучения.

4. Библиотека ENCOG является робастной. Она очень хорошо спроектирована, поддерживает работу с несколькими процессорами/потоками для ускорения нейросетевых расчетов.

5. На текущий момент ECNOG предоставляет возможность реализовать большинство нейросетевых архитектур, алгоритмов обучения, применять различные активационные функции и использовать встроенные генераторы случайных чисел.

Из библиотеки компонентов ENCOG в данной работе будут использоваться методы BasicNetwork() и BasicLayer() предназначенные для программной реализации скрытых слоёв ИНС.

2.3 Представление входных данных

Эффективность обучения ИНС зависит от объёма обучающей выборки. В разрабатываемом программном комплексе на первой эпохе обучения ИНС

будет рассматривать лишь 10 первых значений предлагаемого временного ряда и после этого делать случайное предположение об одиннадцатом.

Эпоха – одна итерация в процессе обучения, включающая предъявление всех примеров из обучающего множества и, возможно, проверку качества обучения на контрольном множестве [19].

Сравнивая прогноз одиннадцатого значения с контрольным множеством, ИНС будет корректировать веса своих нейронов учитывая рассчитанную ошибку прогнозирования. Чем больше значений из обучающего множества рассмотрит ИНС и чем больше эпох она проделает, тем более точными будут прогнозируемые значения.

Входные и выходные данные для программного комплекса будут представлены в формате .csv. CSV (от англ. Comma-Separated Values) — текстовый формат, предназначенный для представления табличных данных. Файлы такого формата можно открывать как в обычном редакторе текстовых файлов, как показано на рисунке 2.2, так и в офисных программных продуктах, например, в Microsoft Excel. Многие приложения (Microsoft Excel, Google Docs, MySQL, Битрикс, PowerShell и др.) поддерживают импорт и экспорт данных в формате .csv, что делает его удобным для работы и позволяет использовать значения для обучающей выборки из разнообразных проектов и программных разработок. Прогнозные значения также могут быть использованы, поскольку программный комплекс должен поддерживать функцию сохранения выходных значений в файл формата.csv.

Формат .csv подразумевает следующие требования для обучающего множества:

1. Каждая строка соответствует одной дате во время которой были сделаны измерения.

2. Первая строка файла должна содержать названия столбцов в латинице.

3. Названия столбцов должны быть разделены запятой. Дата и последующие значения должны быть разделены запятой. Значения во всех столбцах должны быть разделены запятой.

4. Каждая строка должна заканчиваться символом переноса строки.

5. Целая и дробная часть в действительных числах должна быть разделена точкой.

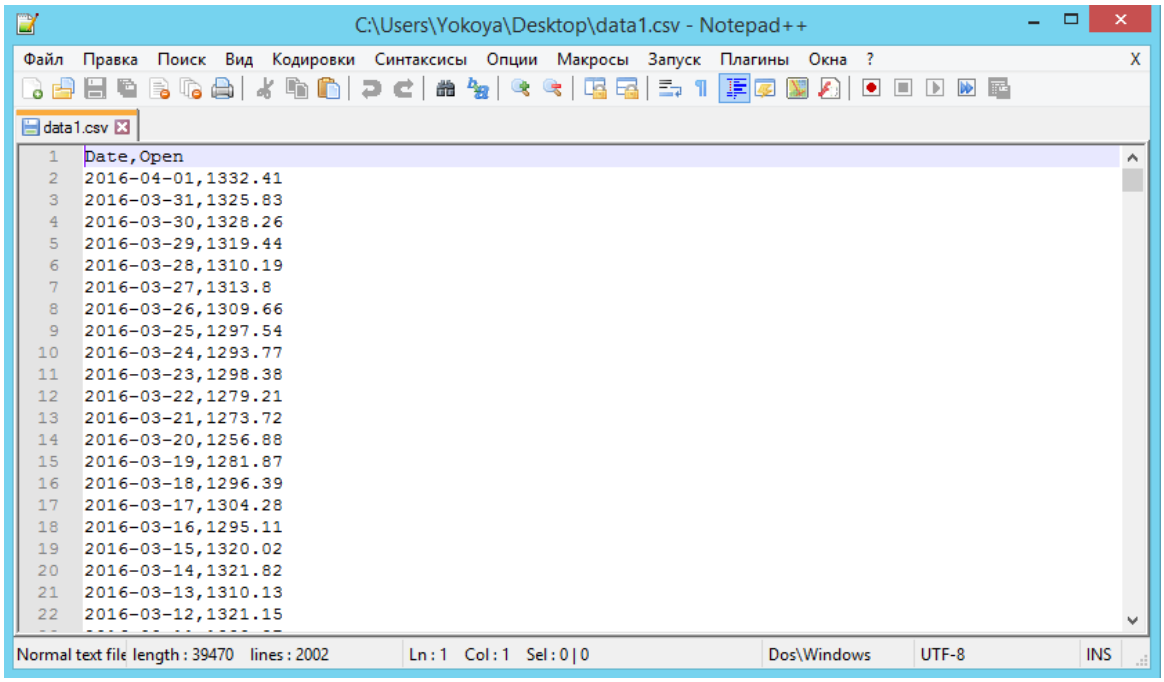


Рисунок 2.2 – Представление данных в формате .csv. Первая обучающая выборка (продажа фруктов) – один столбец со значениями

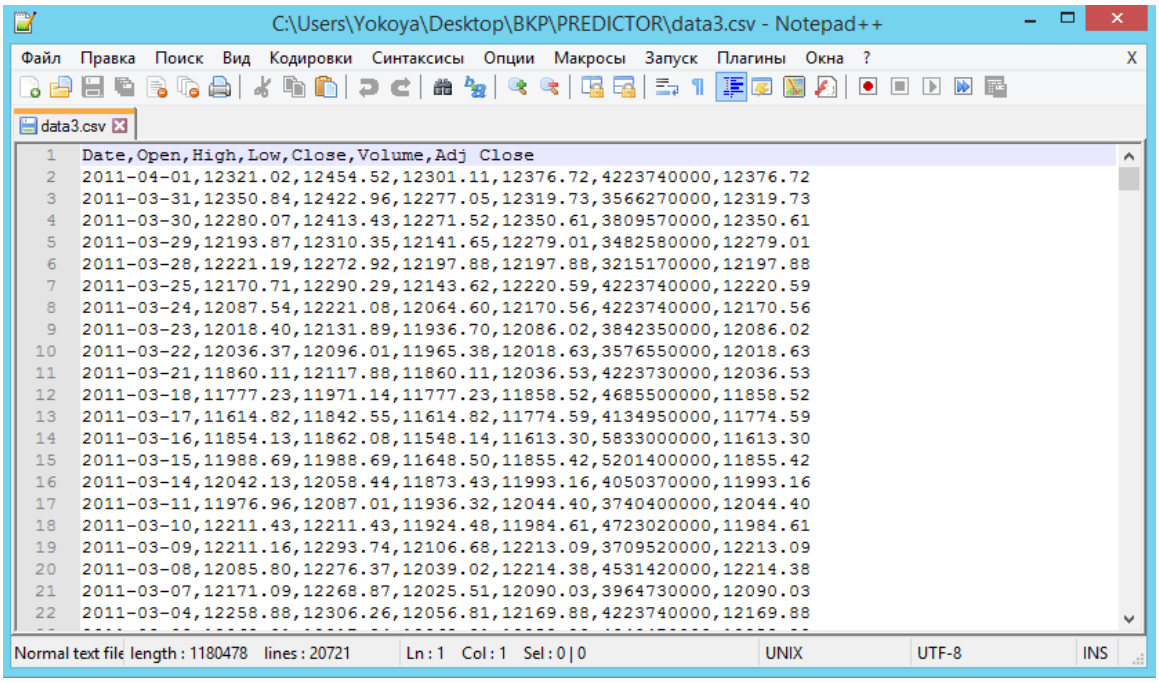


Рисунок 2.3 – Представление данных в формате .csv. Третья обучающая выборка (индекс Dow-Jones) – шесть столбцов со значениями.

Для более эффективного тестирования программного комплекса, в качестве входных данных для ИНС были выбраны 4 массива реальных данных различного типа с идентичными периодами измерений.

1. Данные о продажах фруктов в продуктовой сети. 2000 ежедневных значений (в килограммах) в периоде с 11 октября 2010 до 1 апреля 2016.

2. Данные о количестве посетителей кинотеатра. 500 ежемесячных значений (в тысячах) в периоде с октября 1974 до марта 2016.

3. Индекс Доу-Джонса. 20700 значений в периоде с 12 июня 1958 года до 1 апреля 2016.

4. Данные о посетителях веб-сайта. 2000 ежедневных значений (в количестве уникальных визитов) в периоде с 11 октября 2010 до 1 апреля 2016.

Дополнительным требованием, относящимся конкретно к разрабатываемому программному комплексу, будет наличие последний по дате значений в одном и том же периоде во всех четырёх массивах данных. Это позволит одновременно для всех выборок обучать сеть и демонстрировать прогнозы.

В том случае, если в предоставленной обучающей выборке к каждой дате относятся несколько значений, в программном коде класса PredictorManager указывается название необходимого столбца, как показано на рисунке 2.4.

```
private const string DATE_HEADER = "date";
private const string VALUE_HEADER = "value";
```

Рисунок 2.4 – Указание в программном коде названий столбцов участвующих в процессе обучения

2.4 Описание классовой структуры программного комплекса

Архитектура программного комплекса состоит из 13 классов. Общая архитектура системы представлена на рисунке 2.5.

Класс Program является главная точкой входа для приложения и запускает класс NeuroPredictor.

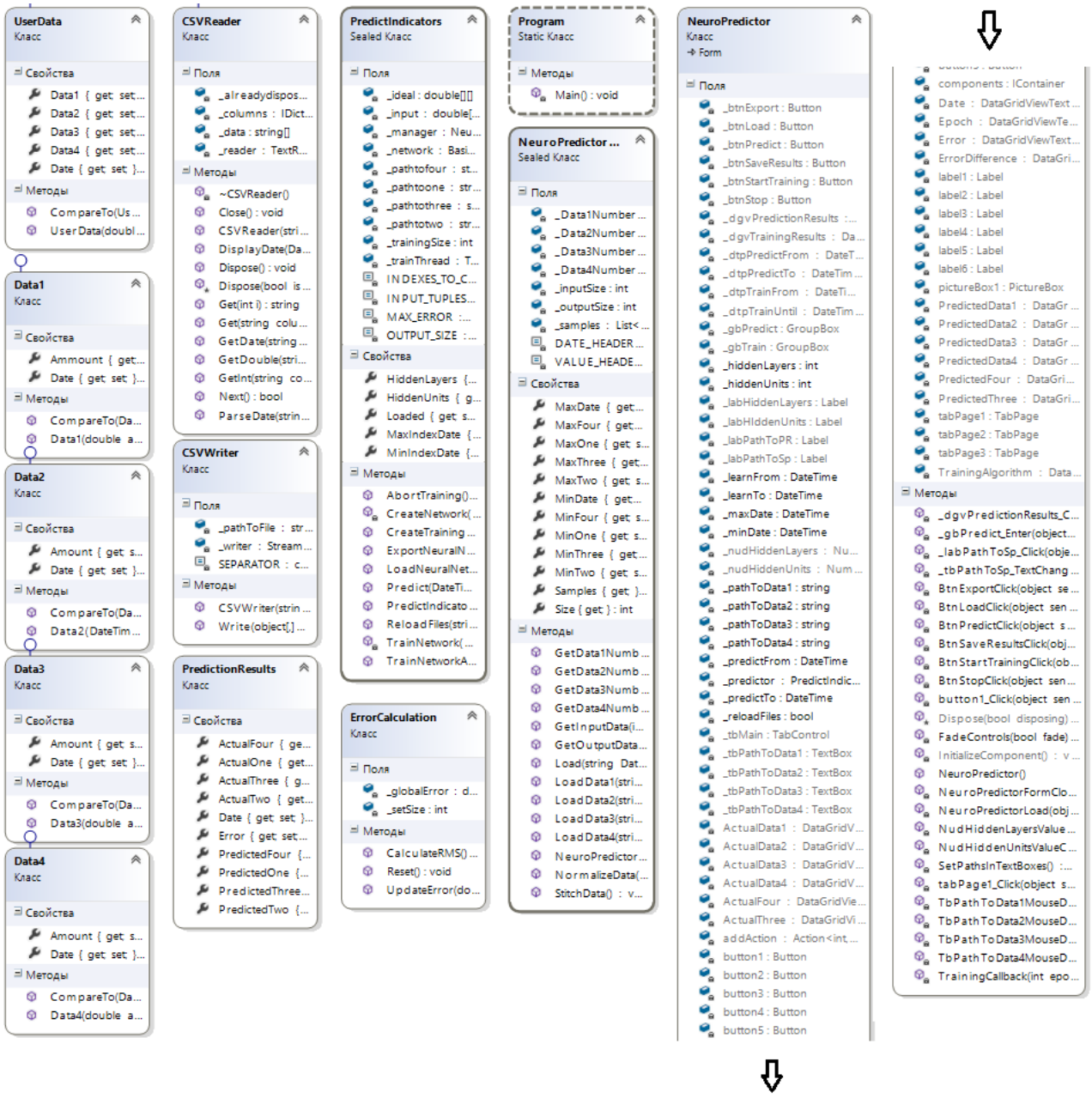


Рисунок 2.5 – Диаграмма классов программного комплекса

Класс NeuroPredictor является главным классом приложения. Содержит следующие важные методы. Содержит значения по умолчанию для всех изменяемых параметров в пользовательском графическом интерфейсе. Отвечает за функционирование всех элементов интерфейса.

Класс NeuroPredictorManager содержит информацию о столбцах обучающей выборки, используемых в процессах обучения и сверки с контрольными значениями. Отвечает за корректную загрузку данных из пользовательских файлов.

Класс PredictIndicators содержит значения количества обучающих выборок и выходных нейронов, размер обучающего окна. Моделирует ИНС с помощью метода CreateNetwork() и обучает её с помощью TrainNetwork(). Обучающая выборка для каждой итерации создаётся в методе CreateTrainingSets(), а окончательный расчёт прогнозных значений (учитывая активационную функцию) происходит в методе Predict(). Также в классе присутствует метод ExportNeuralNetwork() отвечающий за экспорт весов нейронов в уже обученной ИНС.

Класс ErrorCalculation и его внутренние методы CalculateRMS() и UpdateError() рассчитывают среднеквадратичную ошибку прогнозирования.

Методы класса CSVReader отвечают за извлечение необходимых для пользователя значений из файла формата .csv. Методы класса отделяют столбцы со значениями друг от друга, извлекают столбец с датой и определяют тип используемых данных. Методы класса CSVWriter записывают прогнозные значения в файл формата .csv.

Класс PredictionResults отвечает за формирование всех дат, прогнозных значений, обучающих значений и значений ошибки в единой информационное поле, демонстрирующееся пользователю на стартовом экране приложения.

Классы Data1, Data2, Data3 и Data4 позволяют настроить извлечение обучающей выборки для каждого из массива предоставляемых данных. Класс UserData отвечает за сбор значений из всех массивов в единый обучающий массив.

2.5. Программная реализация искусственной нейронной сети

2.5.1 Обоснование выбора нейросетевой архитектуры

Выбор архитектуры ИНС должен обуславливаться типом решаемой задачи. Для задач нейросетевого прогнозировании временных рядов лучше всего подходят архитектуры типа многослойный персептрон (МП) и сеть типа радиально-базисной функции (РБФ).

На сегодняшний день многослойный персептрон – одна из самых используемых нейросетей. Одно из главных преимуществ многослойного персептрона, это возможность решать алгоритмически неразрешимые задачи или задачи, для которых алгоритмическое решение неизвестно, но для которых возможно составить репрезентативный набор примеров с известными решениями. При обучении нейросеть, за счёт своего внутреннего строения, выявляет закономерности в связи входных и выходных образов, тем самым как бы «обобщает» полученный на обучающей выборке опыт. В этой способности к обобщению и состоит основа привлекательности многослойного персептрона. Исследователь может сам и не знать какова зависимость между входными и выходными образами, достаточно иметь большой набор векторов, для которых известен ожидаемый выход.

Многослойными персептронами называют ИНС прямого распространения. В этой ИНС нейроны размещаются последовательными группами, называемыми слоями. Входной сигнал в таких сетях распространяется в прямом направлении, от слоя к слою. Многослойный персептрон в общем представлении состоит из следующих элементов:

- множества входных узлов, которые образуют входной слой;
- одного или нескольких скрытых слоев вычислительных нейронов;
- одного выходного слоя нейронов.

Нейроны в каждом из слоев независимы друг от друга, однако каждый из нейронов связан исходящими связями с каждым нейроном следующего слоя. Таким образом, каждый из нейронов выходного и скрытых слоев принимает входящие сигналы от нейронов предыдущего слоя, как было указано на рисунке 1.2.

Радиально-базисные сети были предложены для аппроксимации функций многих переменных. С помощью радиально-базисных функций можно сколь угодно точно аппроксимировать заданную функцию. Как и многослойный

персептрон, радиально-базисная сеть является универсальным аппроксиматором.

Структура РБФ соответствует сети прямого распространения первого порядка.

Информация об образах передается с входного слоя на скрытый, являющийся шаблонным и содержащий ρ нейронов. Каждый нейрон шаблонного слоя, получая полную информацию о входных сигналах x, вычисляет функцию по формуле 2.1.

$$f_i(x) = f\left((x - c_i)^T R^{-1}(x - c_i)\right), i = \overline{1, p}, \quad (2.1)$$

где вектор входных сигналов $(Nx1)$; c_i – вектор центров $(Nx1)$; R – весовая матрица.

Особенностью данных сетей является наличие радиально-симметричного шаблонного слоя, в котором анализируется расстояние $(x - c_i)^T R^{-1}(x - c_i)$ между входным вектором и центром, представленным в виде вектора во входном пространстве. Вектор центров определяется по обучающей выборке и сохраняется в пространстве весов от входного слоя к слою шаблонов.

В ходе исследования установлено, что радиальные базисные сети обладают рядом преимуществ перед сетями типа многослойный персептрон. Во-первых, они моделируют произвольную нелинейную функцию с помощью одного промежуточного слоя. Тем самым отпадает вопрос о числе слоев. Во-вторых, параметры линейной комбинации в выходном слое можно полностью оптимизировать с помощью известных методов моделирования, которые не испытывают трудностей с локальными минимумами, мешающими при обучении МП. Поэтому сеть РБФ обучается на порядок быстрее МП.

С другой стороны, до того, как применять линейную оптимизацию в выходном слое сети РБФ, необходимо определить число радиальных элементов, положение их центров и величины отклонений. Для устранения этой проблемы предлагается использовать автоматизированный конструктор сети, который выполняет за пользователя основные эксперименты с сетью.

Другие отличия работы РБФ от МП связаны с различным представлением пространства модели: «групповым» в РБФ и «плоскостным» в МП. Опыт показывает, что для правильного моделирования типичной функции, сеть РБФ требует несколько большего числа элементов. Следовательно, модель, основанная на РБФ, будет работать медленнее и потребует больше памяти, чем соответствующий МП (однако она гораздо быстрее обучается, а в некоторых случаях это важнее).

Скорость работы и затраты памяти являются важным критерием при разработке программного комплекса, работающего с большими массивами данных, поэтому для задачи прогнозирования значений временного ряда, будет использоваться многослойная сеть прямого распространения, или МП.

2.5.2 Описание математической модели программируемого нейрона

Искусственный нейрон – узел искусственной нейронной сети, являющийся упрощённой моделью естественного нейрона. Математически, искусственный нейрон обычно представляют, как некоторую нелинейную функцию от единственного аргумента — линейной комбинации всех входных сигналов [20]. Математическая модель нейрона представлена на рисунке 2.2.

Можно увидеть, что существует 3 основных элемента внутри нейронной модели:

1. Набор синапсов или соединительных звеньев, каждое из которых характеризуется «весами» $w_{k0}, w_{k1}, \dots, w_{km}$. Весовые коэффициенты представляют собой данные, которые ИНС получила в результате обучения.

2. Сумматор для суммирования входных сигналов, взвешенных после прохождения по синапсам нейрона: $v_k = \sum(w_{kj}x_j + b_k)$. Иначе говоря, входной сигнал X умножается на вес W и суммируется в сумматоре с другими входными значениями. Результат этого суммирования V отправляется на вход функции активации.

3. Функция активации для ограничения выходного сигнала нейрона: $Y_k = \varphi(x)$. Эта функция генерирует выходной сигнал в соответствии с числами, полученными в сумматоре. При этом выходной сигнал каждого нейрона должен быть определен следующим образом: $Y_k = \varphi\left(\sum(w_{kj}x_j + b_k)\right)$. При использовании алгоритма обучения обратного распространения ошибки необходимо, чтобы функция активации была дифференцируема. Это требование исходит из того, что, поскольку этот метод требует вычисления градиента функции ошибки на каждом шаге итерации, мы должны гарантировать непрерывность и дифференцируемость функции ошибки.

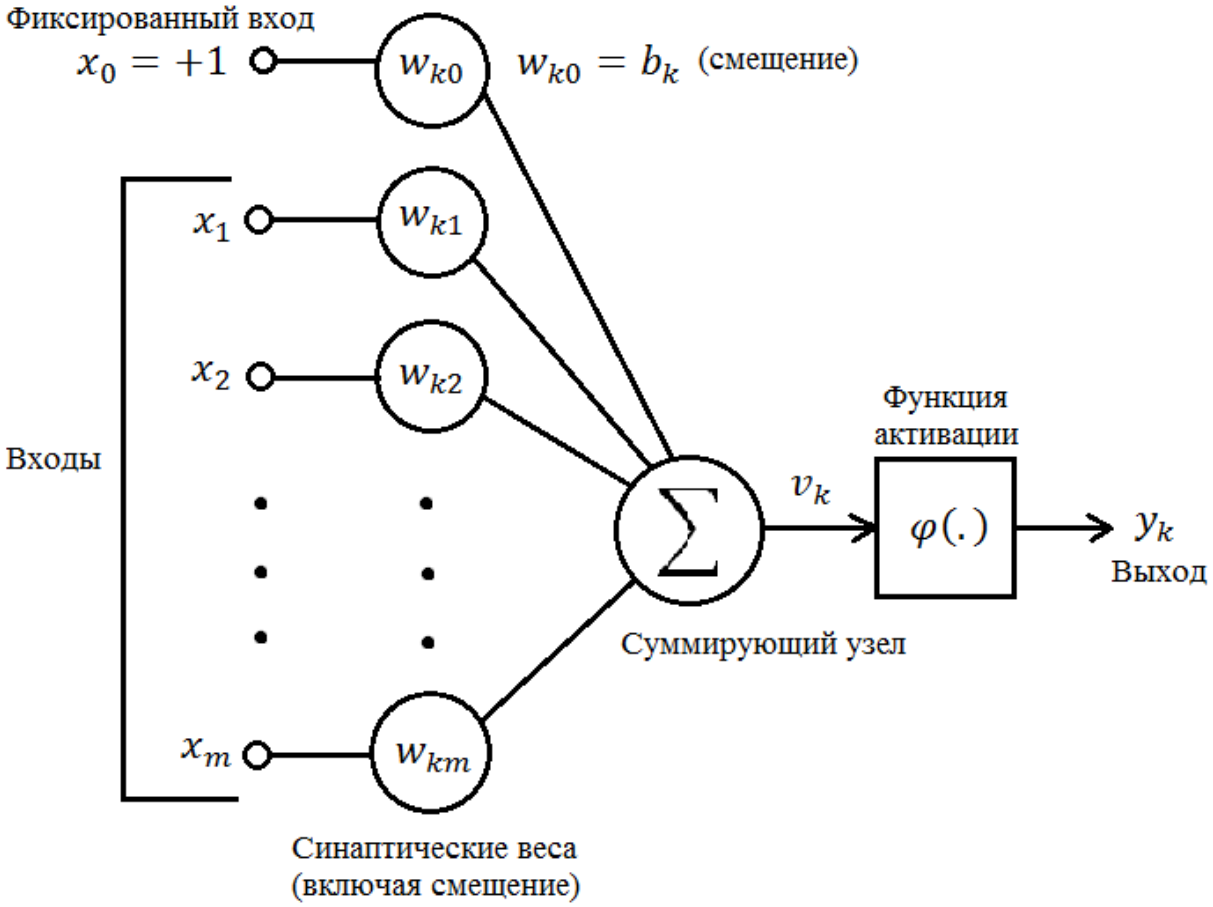


Рисунок 2.6 – Математическая модель нейрона

Обычно используемая в таких случаях нелинейная функция сигмоиды определяется логистической функцией: $\varphi(v) = \frac{1}{1 + \exp(-\alpha v)}$, где α - параметр

наклона сигмовидной функции. Варьируя значение параметра α, получаем различные виды сигмовидной функции, которые показаны на рисунке 2.6.

ИНС работает с числами с плавающей точкой, поэтому для объявления и хранения любых числовых данных в языке C# будет использоваться тип double.

2.5.3. Программная реализация слоёв искусственной нейронной сети

Нейронная сеть прямого распространения состоит из нейронов, которые сгруппированы в слоях. Должно присутствовать как минимум 2 слоя: входной слой, содержащий входные нейроны и выходной слой с нейронами выходного слоя. Между входным и выходным слоями также могут присутствовать и скрытые слои. Входной слой может быть обычным массивом из чисел типа double, выходной слой может содержать один или более нейронов, которые (в общем случае) также представляют собой одномерный массив из чисел типа double.

Каждая связь имеет свой вес, который также представляет собой число типа double, и функцию активации с порогом, который отвечает за активацию нейрона и прохождение информации к следующему нейрону. По этой причине такая сеть называется сетью прямого распространения – информация с выходов активированных нейронов проходит вперед к следующему слою нейронов.

Многослойные нелинейные нейронные сети позволяют формировать более сложные связи между входами и выходами, чем однослойные линейные. Доказано, что двухслойная нейронная сеть с одним скрытым слоем может быть обучена аппроксимировать с произвольной точностью любую непрерывную функцию [5]. Данное утверждение является основой для аппроксимации и экстраполяции функций при помощи многослойных нейронных сетей.

Проблемы, которые требуют более чем двух скрытых слоёв встречаются достаточно редко и поэтому, такие сети нет смысла использовать в задачах о принятии решений. Несмотря на то, что скрытые слои не взаимодействуют непосредственно с внешней средой, они оказывают огромное влияние на

конечный результат работы сети. Поэтому, количество нейронов в скрытых слоях должно обдумываться на стадии проектирования. Использование слишком малого количества нейронов приведет к так называемому «недообучению», которое происходит, когда скрытые слои не способны адекватно обнаружить факторы, влияющие на сигналы в сложном наборе данных. Проблема «недообучения» также может возникнуть, когда ИНС имеет так много мощностей для обработки информации, что ограниченное количество информации, содержащейся в обучающей выборке недостаточно, чтобы обучить все нейроны в скрытых слоях. Есть много практических методов для определения правильного количества нейронов, вот лишь некоторые из них [17]:

1. Количество скрытых нейронов должно быть равно числу в диапазоне от размера входного слоя до размера выходного.

2. Количество скрытых нейронов должно быть равно двум-третьим размера входного слоя плюс размер выходного слоя.

3. Количество скрытых нейронов должно быть меньше, чем удвоенное количество элементов во входном слое.

Разрабатываемая ИНС будет работать с 10-ю последовательными значениями каждого из 4-х типов исходных данных. Итого на вход будут отправляться 40 отсортированных по дате числовых значений, которые будут описывать 10 дней каждого массива реальных данных. Сеть будет пытаться предсказать 11-е значение, которое будет соответствовать следующему дню в строке каждого типа данных. Последний слой сети будут составлять 4 выходных нейрона, прогнозирующих значение 11-го дня.

Основываясь на вышесказанном, можно предложить для прогнозирования временного ряда ИНС, структура которой включает входной слой и два слоя нейронных элементов. Входной слой нейронной сети выполняет распределительные функции и содержит 40 нейронов. Скрытые нейронные слои производят обработку информации. Их количество может быть изменено в пользовательском интерфейсе программы. Выходной нейронный

слой служит как для обработки информации от предыдущих слоев, так и для выдачи результатов и содержит 4 выходных нейрона.

Окончательно определившись с количеством слоёв, нейронов и функцией активации, можно приступить к кодированию метода, который отвечает за создание новой ИНС. Реализация метода показана на рисунке 2.7.

```
        private const int INDEXES_TO_CONSIDER = 4;
        private const int INPUT_TUPLES = 10;
        private const int OUTPUT_SIZE = 4;
        private const double MAX_ERROR = 0.00005;
        // Создание новой нейросети
        private void CreateNetwork(int hiddenUnits, int hidden Layers )
        {
            _network = new BasicNetwork {Name = “Predictor”, Description
= “Network for prediction analysis”};
            _network.AddLayer(new BasicLayer(INPUT_TUPLES *
INDEXES_TO_CONSIDER));                                  /*Вход*/
            for (int  = 0; i < hidden Layers ; i++)
                _network.AddLayer(new BasicLayer(new ActivationTANH(),
true, hiddenUnits));                        /*Скрытый слой*/
            _network.AddLayer(new BasicLayer(new ActivationTANH(), true,
OUTPUT_SIZE));                             /*Вывод нейросети*/
            _network.Structure.FinalizeStructure();
/*Выходной слой*/
            _network.Reset();
/*Заполнение случайными числами*/
             }
```

Рисунок 2.7 – Программная реализация слоёв ИНС

BasicNetwork – вызывает метод библиотеки компонентов ENCOG, создающий нелинейную ИНС. BasicLayer – вызывает метод создающий слой ИНС. Метод ActivationTann() – это функция активации нейрона. Значения hiddenUnits и hidden Layers могут быть изменены в графическом интерфейсе программного комплекса. Значения переменных INPUT_TUPLES, INDEXES_TO_CONSIDER и OUTPUT_SIZE прописываются в программном коде.

2.5.4 Программная реализация функции активации «сигмоида»

Функция активации «сигмоида» генерирует выходной сигнал в соответствии с числами полученными в сумматоре. При этом выходной сигнал каждого нейрона должен быть определен следующим образом:

$Y_k = \varphi(\sum [(w_{kj} x_j + b_k)]$. При использовании алгоритма обучения обратного распространения ошибки необходимо, чтобы функция активации была дифференцируема. Это требование исходит из того, что, поскольку этот метод требует вычисления градиента функции ошибки на каждом шаге итерации, мы должны гарантировать непрерывность и дифференцируемость функции ошибки. Обычно используемая в таких случаях нелинейная функция сигмоиды определяется логистической функцией: $\varphi(v) = \frac{1}{1 + \exp(-\alpha v)}$., где α - параметр наклона сигмовидной функции. Варьируя значение параметра α, получаем различные виды сигмовидной функции, которые показаны на рисунке 2.8 [12].

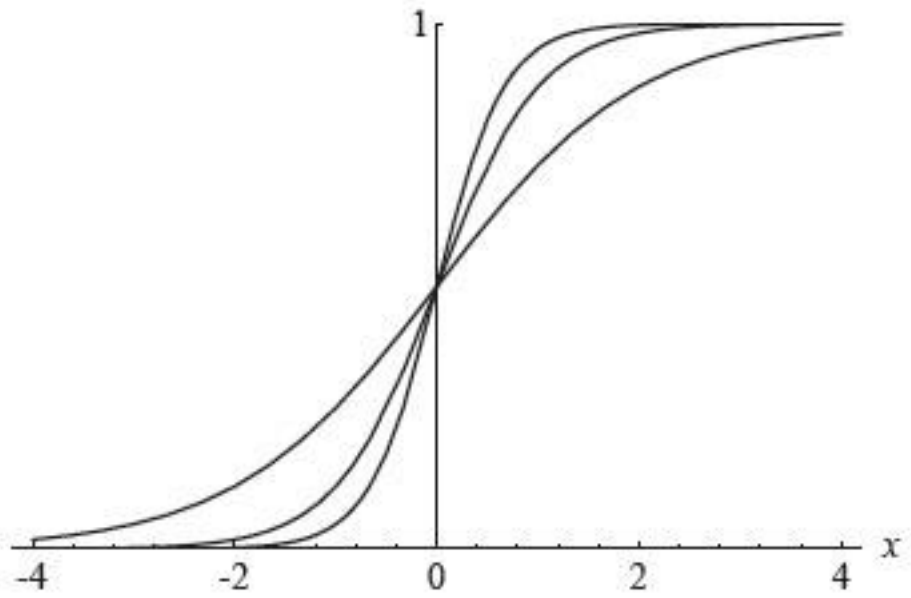


Рисунок 2.8 – Функция активации «сигмоида»

Программная реализация алгоритма активационной функции нейрона показана на рисунке 2.9.

```
public virtual void ActivationFunction(double[] x, int start,
                                       int size)
{
    for (38nti  = start; i < start + size; i++)
    {
        x[i] = 2.0 / (1.0 + BoundMath.Exp(-2.0 * x[i])) - 1.0;
    }
}

public virtual double DerivativeFunction(double b, double a)
{
    return (1.0d - a * a);}
```

Рисунок 2.9 – Программная реализация алгоритма активационной функции нейрона

2.5.5 Выбор метода обучения и программная реализация

Для обучения необходим массив обучающих примеров. Количество примеров должно быть достаточно большим – по некоторым расчетам, в 10-15 раз больше числа нейронов в сети. Примеры предъявляются ИНС, при этом веса связей внутри нее постепенно изменяются, с тем, чтобы реальный выходной сигнал был как можно ближе к ожидаемому значению выходного фактора. Один цикл предъявления всех учебных образцов называется эпохой. Обычно требуется несколько тысяч итераций, чтобы обучить ИНС, но на современных компьютерах такое обучение занимает несколько минут.

Часть примеров не участвует в обучении, а выделяется в так называемое тестовое множество. На каждой эпохе работа сети проверяется на тестовом множестве. Таким образом тестируется способность ИНС к обобщению: возможности распространить выявленную закономерность к данным, не участвующим в обучении.

Обучение ИНС заканчивается, когда достигнуто заданное значение средней (или минимальной) ошибки, когда сеть исчерпала возможности обучения или же когда пройдено определенное число эпох. После этого веса связей фиксируются, и сеть может использоваться в рабочем режиме. Теперь, если в качестве входных сигналов сети указать параметры входного множества, значение на выходе будет представлять его прогноз, рассчитанный на основе выявленной закономерности.

Согласно вышеизложенного материала можно увидеть главное отличие ИНС от экспертных систем. Если в экспертной системе знания извлекаются из опыта специалистов, то ИНС сама накапливает опыт на основе просмотра набора аналогичных примеров, и фиксирует его в виде набора весов связей.

Не всегда ИНС достигает хороших результатов обучения и обобщения. Среди возможных причин можно выделить следующие:

– неудачно выбрана архитектура сети (слишком много или слишком мало нейронов в скрытых слоях);

– недостаточно примеров для обучения;

– влияющие факторы выделены неудачно: в число входных параметров не включен один или несколько факторов, в наибольшей мере влияющий на значение выходных показателей;

– искомой зависимости не существует; обучающие примеры являются уникальными, аналогия между ними отсутствует.

Обучающие алгоритмы могут быть классифицированы как алгоритмы обучения с учителем и обучения без учителя. В первом случае существует учитель, который предъявляет входные образы сети, сравнивает результирующие выходы с требуемыми, а затем настраивает веса сети таким образом, чтобы уменьшить различия. Трудно представить такой обучающий механизм в биологических системах; следовательно, хотя данный подход привел к большим успехам при решении прикладных задач, он отвергается теми исследователями, кто полагает, что ИНС обязательно должны использовать те же механизмы, что и человеческий мозг [11].

Во втором случае обучение проводится без учителя: при предъявлении входных образов сеть самоорганизуется, настраивая свои веса согласно определенному алгоритму. Требуемый выход в процессе обучения не указан, поэтому результаты определения возбуждающих образов для конкретных нейронов непредсказуемы. При этом, однако, сеть организуется в форме, отражающей существенные характеристики обучающего набора. Например,

входные образы могут быть классифицированы согласно степени их сходства так, что образы одного класса активизируют один и тот же выходной нейрон.

Разрабатываемая ИНС на основе многослойного персептрона является двухуровневой сетью. Она использует алгоритм обучения с учителем; другими словами, обучающая выборка состоит из множества входных векторов, для каждого из которых указан свой требуемый вектор цели.

Алгоритм обратного распространения ошибки является одним из методов обучения многослойных персептронов. Обучение алгоритмом обратного распространения ошибки предполагает два прохода по всем слоям сети: прямого и обратного. При прямом проходе входной вектор подается на входной слой ИНС, после чего распространяется по сети от слоя к слою. В результате генерируется набор выходных сигналов, который и является фактической реакцией сети на данный входной образ. Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода все синаптические веса настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в направлении, обратном направлению синаптических связей.

Одним из серьезных недостатков алгоритма с обратным распространением ошибки, используемого для обучения многослойных нейронных сетей, является слишком долгий процесс обучения, что делает неприменимым использование данного алгоритма для широкого круга задач, которые требуют быстрого решения. Одним из альтернативных решений является алгоритм Resilient Propagation (RProp) который был предложен М. Ридмиллером и Г. Брауном.

В отличие от стандартного алгоритма обратного распространения ошибки, RProp использует только знаки частных производных для подстройки весовых коэффициентов. Алгоритм использует так называемое «обучение по

эпохам», когда коррекция весов происходит после предъявления сети всех примеров из обучающей выборки.

Для определения величины коррекции используется формула 2.2.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+}\Delta_{ij}^{(t)}{}_{ij}^{(t)}, \dfrac{\varphi E^{(t)}}{\varphi w_{ij}} \dfrac{\varphi E^{(t-1)}}{\varphi w_{ij}} > 0 \\ \eta^{-}\Delta_{ij}^{(t)}, \dfrac{\varphi E^{(t)}}{\varphi w_{ij}} \dfrac{\varphi E^{(t-1)}}{\varphi w_{ij}} < 0 \end{cases} \quad (2.2)$$

Если на текущем шаге частная производная по соответствующему весу w_{ij} поменяла свой знак, то это говорит о том, что последнее изменение было большим, и алгоритм проскочил локальный минимум и, следовательно, величину изменения необходимо уменьшить на η и вернуть предыдущее значение весового коэффициента: другими словами необходимо произвести «откат» [18], демонстрируемый на формуле 2.3.

$$\Delta w_{ij}(t) = \Delta w_{ij}(t) - \Delta_{ij}^{(t-1)} \quad (2.3)$$

Если знак частной производной не изменился, то нужно увеличить величину коррекции на η^{+} для достижения более быстрой сходимости. Зафиксировав множители η^{-} и η^{+}, можно отказаться от глобальных параметров настройки ИНС, что также можно рассматривать как преимущество рассматриваемого алгоритма перед стандартным алгоритмом обратного распространения ошибки.

Поскольку алгоритм обучения RProp сходится в 4-5 раз быстрее, чем стандартный алгоритм, то именно был выбран для реализации в разрабатываемом программном комплексе. Программная реализация обучения ИНС представлена на рисунке 2.10.

```
        private void TrainNetwork ( DateTime trainFrom, DateTime trainTo,
TrainingStatus status )
        {
            if ( _input == null || _ideal == null )
                CreateTrainingSets ( trainFrom, trainTo ) ;
            _trainThread = Thread . CurrentThread;
            int epoch = 1;
            ITrain train = null;
            try
            {
                var trainSet = new BasicNeuralDataSet ( _input, _ideal );
                train = new ResilientPropagation ( _network, trainSet );
                double error;
                do
                {
                    train . Iteration (  );
                    error = train . Error;
                    if  ( status != null )
                        status . Invoke ( epoch, error, TrainingAlgorithm
. Resilient ) ;
                    epoch++;
                } while  ( error > MAX_ERROR );
            }
            catch  ( ThreadAbortException )  { _trainThread = null; }
            finally
            {
                train . FinishTraining (  );
            }
            _trainThread = null;
        }
```

Рисунок 2.10 – Программная реализация метода обучения RProp

2.6 Программная реализация выделения трендовой компоненты временного ряда

На входе функция получает таблицу значений функций — матрицу, в первом столбце которой содержатся значения x, во втором, соответственно, y, а также значение степенного базиса.

Сначала выделяется память под матрицу, в которую будут записаны коэффициенты для решения системы линейных уравнений. Затем, собственно, составляем матрицу — в sumA записываются значения коэффициентов a_{ij}, в sumB — b_i, все по формуле, указанной в первой главе бакалаврской работы.

Для решения составленной системы линейных алгебраических уравнений в моей программе используется метод Гаусса, программная реализация которого представлена на рисунке 2.11.

```
Private double[,] MakeSystem(double[,] xyTable, int basis)
        {
            double[,] matrix = new double[basis, basis + 1];
            for (45nti  = 0; i < basis; i++)
            {
                for (int j = 0; j < basis; j++)
                {
                    matrix[i, j] = 0;
                }
            }
            for (45nti  = 0; i < basis; i++)
            {
                for (int j = 0; j < basis; j++)
                {
                    double sumA = 0, sumB = 0;
                    for (int k = 0; k < xyTable.Length / 2; k++)
                    {
                        sumA += Math.Pow(xyTable[0, k], i) *
Math.Pow(xyTable[0, k], j);
                        sumB += xyTable[1, k] * Math.Pow(xyTable[0, k],
i);
                    }
                    matrix[i, j] = sumA;
                    matrix[i, basis] = sumB;
                }
            }
            return matrix;
        }
```

Рисунок 2.11 – Программная реализация выделения трендовой компоненты временного ряда

2.7 Программная реализация расчёта ошибки прогнозирования

RMSE – среднеквадратическая ошибка прогноза применяется в ситуациях, когда нам надо подчеркнуть большие ошибки и выбрать модель, которая дает меньше больших ошибок прогноза.

Грубые ошибки становятся заметнее за счет того, что ошибку прогноза мы возводим в квадрат. И модель, которая дает нам меньшее значение среднеквадратической ошибки, можно сказать, что что у этой модели меньше грубых ошибок.

Программная реализация подсчёта среднеквадратичной ошибки показана на рисунке 2.12.

```
        // Возвращает среднеквадратичную ошибку используемую при
    обучении.
        Public double CalculateRMS()
        {
            return Math.Sqrt(_globalError/(_setSize));
        }

        // Обнуляет значение ошибки.
        Public void Reset()
        {
            _globalError = _setSize = 0;
        }

        // Вызывается для каждого значения, которое должно быть
проверено.
        public void UpdateError(double[] actual, double[] ideal)
        {
            for (46nti  = 0; i < actual.Length; i++)
            {
                double delta = ideal[i] - actual[i];
                _globalError += delta * delta;
            }
            _setSize += ideal.Length;}
```

Рисунок 2.12 – Программная реализация подсчёта среднеквадратичной ошибки прогноза

Расчёт среднеквадратической ошибки происходит в 3 этапа:

1. Рассчитываем ошибку для каждого значения модели.
2. Возводим ошибку в квадрат.
3. Рассчитываем среднее по квадрату ошибки, т.е. среднеквадратическую ошибку.

2.8 Описание графического интерфейса

Программный комплекс состоит из трёх модулей, разделённых на три вкладки расположенных в интерфейсе оконного приложения. Окно программы и основные элементы управления создавались с помощью интерфейса программирования приложений Windows Forms. Программная реализация всех модулей и функций программного комплекса представлена в Приложении А.

Вкладка выводящая прогнозные значения и их сравнение с настоящими значениями используя математический метод поиска ошибки прогнозирования, представлена на рисунке 2.13.

Элементы в нижней части экрана являются общими для всех вкладок. Два первых элемента отображают единичные числовые значения количества скрытых слоёв и нейронов. С помощью кнопок «вверх-вниз» пользователь сможет изменить эти значения на свои собственные. В элементах «путь» необходимо указать пути в файловой системе ко всем четырём массивам обучающих данных. В верхней части экрана расположены кнопки переключения вкладок-модулей и функционал выбора «горизонта прогнозирования».

Горизонт прогнозирования – крайний срок, для которого прогноз действителен с заданной точностью [7]. В некоторых работах этот термин трактуется иначе: как промежуток времени, на который рассчитывается прогноз.

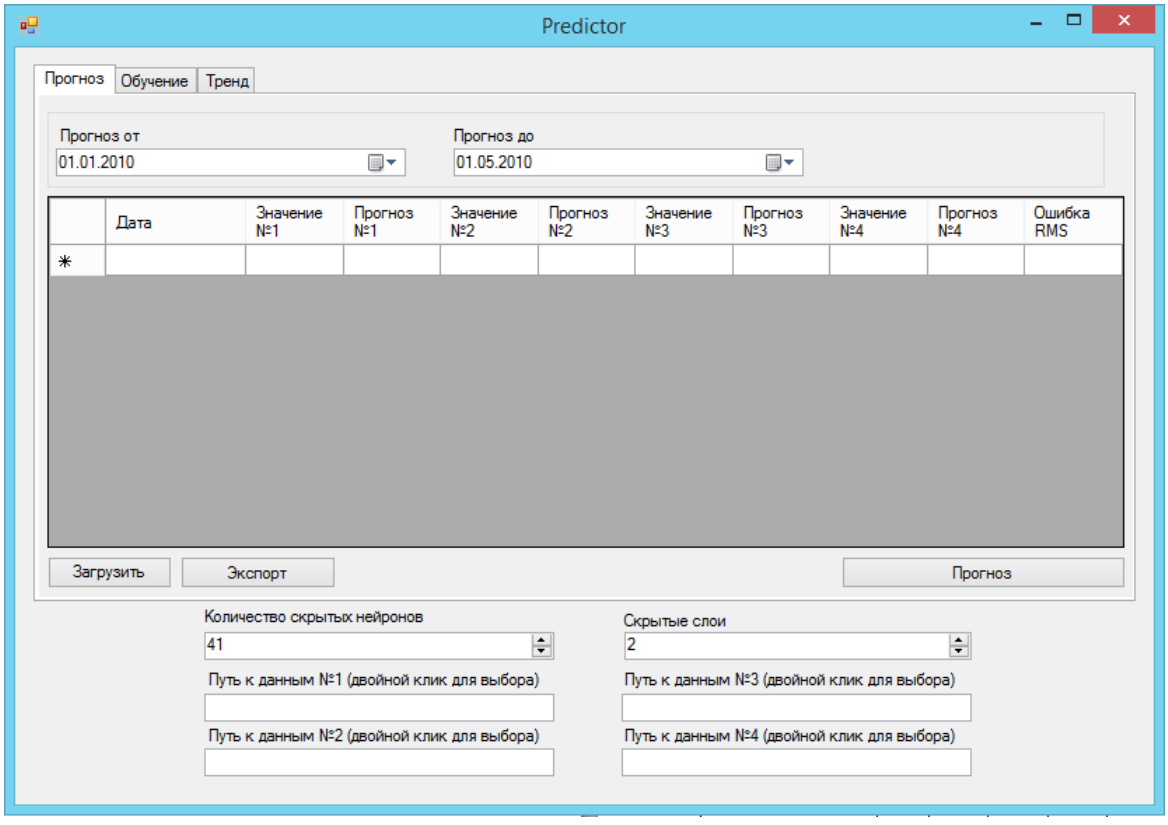


Рисунок 2.13 – Графический интерфейс пользователя. Вкладка «Прогноз»

В центре располагается окно, демонстрирующее прогнозные и контрольные значения, привязанные к определённой дате. Последний столбец рассчитывает общую для всех ошибку прогнозирования RMS. Кнопка

«Загрузить» позволяет внести в память программы значения заранее обученной ИНС. Кнопка Экспорт» позволяет экспортировать данные в .csv файлы. Кнопка «Прогноз» рассчитывает все прогнозные значения для выбранного горизонта прогнозирования.

На рисунке 2.14 изображён интерфейс вкладки «Обучение». В верхней части выбирается период используемых в обучающей выборке значений. Ниже происходит пошаговая демонстрация процесса обучения ИНС. Обучение заканчивается после завершения последней возможной эпохи или после нажатия кнопки «Стоп». Кнопка «Сохранить» позволяет сохранить файл с данными обученной ИНС, чтобы иметь возможность запускать процесс прогнозирования идентичных данных без повторного обучения.

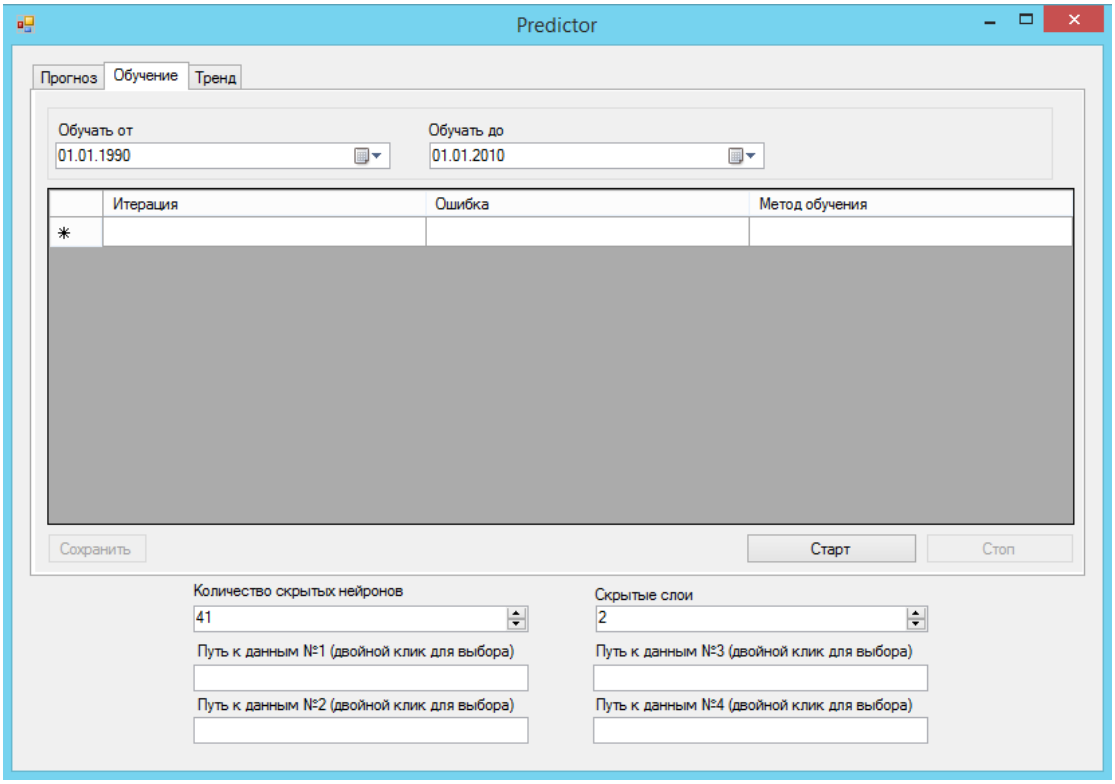


Рисунок 2.14 – Графический интерфейс пользователя. Вкладка «Обучение»

Интерфейс вкладки «Тренд» изображён на рисунке 2.15. Нажатия кнопок «Тренд 1», «Тренд 2», «Тренд 3» и «Тренд 4» вызывают графическое представление трендовой компоненты временного ряда для обучающих данных. Кнопка «Очистить» очищает поле с графиками.

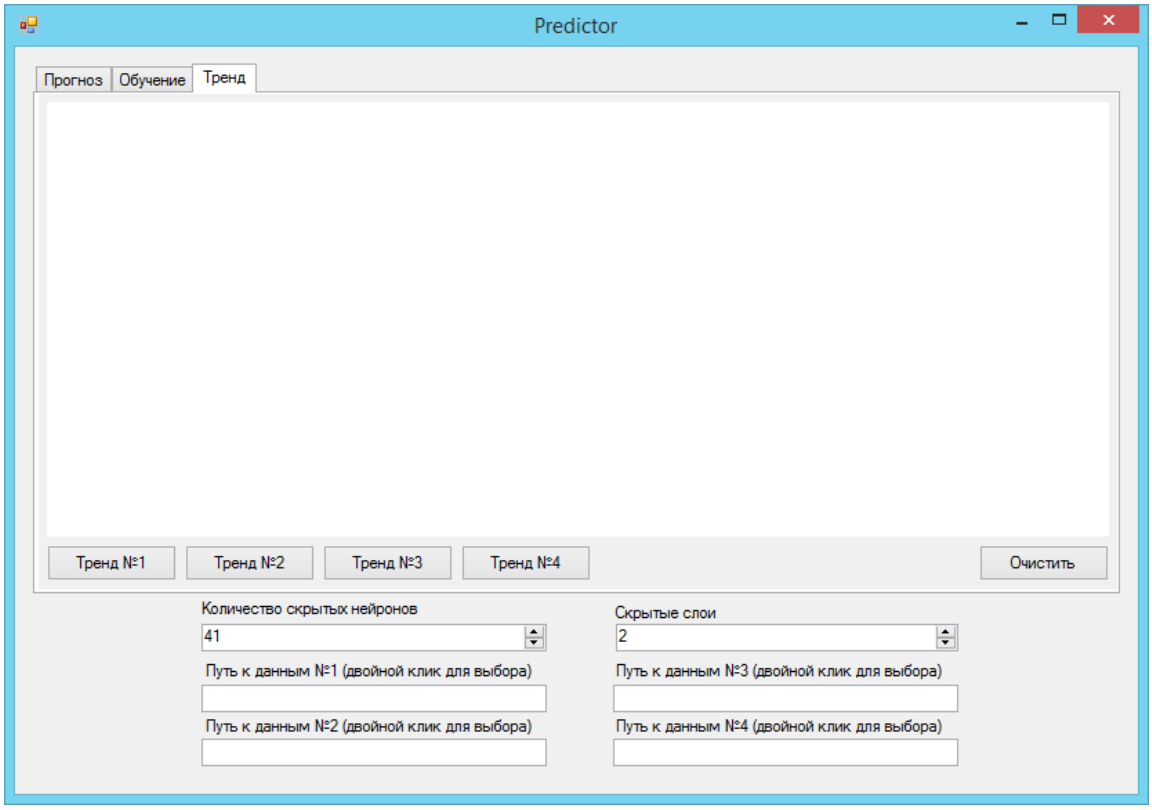


Рисунок 2.15 – Графический интерфейс пользователя. Вкладка «Тренд»

Глава 3 Обучение и тестирование программного комплекса

3.1 Тестирование модуля «Обучение» и отбор оптимальных моделей искусственных нейронных сетей

Хотя нейросетевые модели являются весьма эффективными в задачах оценки, поиск подходящих параметров для проведения обучения может стать весьма трудоёмкой задачей. Исходя из популярных методов выбора количества нейронов в скрытых слоях, определим 5 различных нейросетевых моделей, которые будут использоваться в ходе серии экспериментальных обучений:

1. Модель МП с двумя скрытыми слоями и 22 нейронами в каждом из них (40-22-22-4). Правило используемое при моделировании: Количество скрытых нейронов должно быть равно числу в диапазоне от размера входного слоя до размера выходного.

2. Модель МП с двумя скрытыми слоями и 30 нейронами в каждом из них (40-30-30-4). Правило используемое при моделировании: Количество скрытых нейронов должно быть равно двум-третьим размера входного слоя плюс размер выходного слоя.

3. Модель МП с двумя скрытыми слоями и 79 нейронами в каждом из них (40-79-79-4). Правило используемое при моделировании: Количество скрытых нейронов должно быть меньше, чем удвоенное количество элементов во входном слое.

4. Модель МП с двумя скрытыми слоями и 41 нейронами в каждом из них (40-41-41-4). Усреднённое количество скрытых нейронов из предыдущих значений.

5. Модель МП с одним скрытым слоем и 41 нейронами в каждом из них (40-41-4). Экспериментальная модель с одним скрытым слоем.

Каждая модель будет обучена при помощи 3000 эпох и обучающей выборки состоящей из значений периода с 1 января 2011 года по 1 января 2016

года. Каждый из четырёх массивов входных данных используемых для обучения ИНС содержит в себе 1826 значений (по одному на каждый день) в обозначенном диапазоне. Исключением являются данные о посетителях кинотеатров, в которых предложено лишь одно значение для каждого месяца. В этом случае значение, например, января 2011 года присваивается каждому январскому дню. Процесс обучения модели ИНС с двумя скрытыми слоями по 41 нейрону изображён на рисунке 3.1. Как только ИНС просмотрит все 1826 значений, она вновь вернётся к самому первому, либо остановит свою работу после достижения минимальной ошибки прогнозирования RSE. Значение минимальной ошибки прогнозирования задано в программном коде и составляет 0.00005.

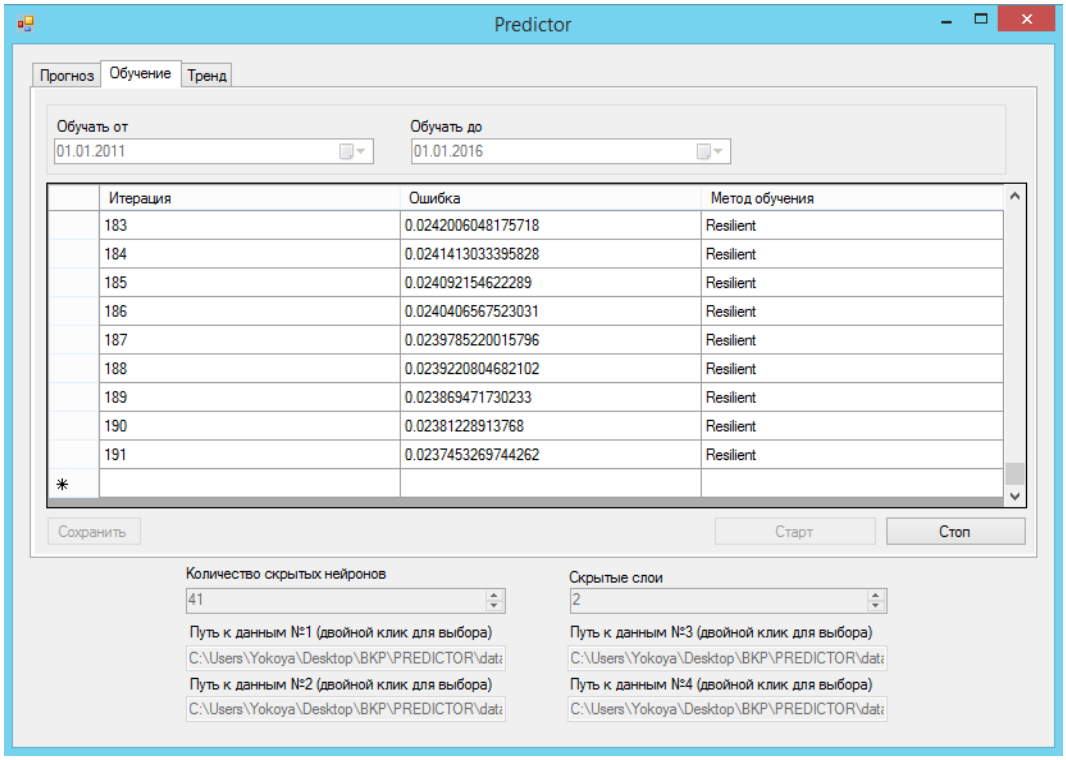


Рисунок 3.1 – Обучение модели 40-41-41-4

Результаты обучения различных моделей представлены в таблице 3.1.

Таблица 3.1 – Среднеквадратичная ошибка на разных этапах обучения различных моделей ИНС

Ошибка RSE	**После 100 эпох**	**После 500 эпох**	**После 1000 эпох**	**После 3000 эпох**
40-22-22-4	0.035772116	0.016635524	0.015123067	0.013572073
40-30-30-4	0.045753455	0.019616916	0.015843094	0.013567889
40-79-79-4	0.037598	0.014555	0.012711	0.011838
40-41-41-4	0.030983138	0.017629552	0.015558595	0.013345895
40-41-4	0.025963	0.018347	0.016269	0.014223

Сравнивая результаты обучения различных моделей, можно сделать вывод, что разница в ошибке прогнозирования почти не отличается для моделей с двумя скрытыми слоями. Тем не менее, если целью прогнозирования являются финансовые данные или данные бизнес-процессов, то важной может стать любая разница в значении ошибки прогнозирования, поскольку более точные данные будут более эффективны в процессе принятия решений.

Длительность обучения модели 40-79-79-4 проходило примерно в 7 раз дольше, чем модели 40-30-30-4, поскольку модель с 79-ю скрытыми нейронами имеет гораздо большее количество связей. При этом данная модель стала предсказывать самые точный результаты по сравнению с остальными моделями примерно с 300-ой эпохи. Несмотря на большое количество скрытых нейронов и связей, сеть не была «недообучена» и доказала справедливость правил построения нейросетевых моделей.

График первых 400 эпох обучения модели 40-79-79-4 изображён на рисунке 3.2. Самые большие значения среднеквадратичной ошибки получаются во время нескольких первых десяток эпох, когда ИНС не имеет достаточного количества данных для обучения. Со временем, прогнозируемые значения стабилизируются и всё больше начинают приближаться к значениям контрольной выборки. График приближения представлен на рисунке 3.3.

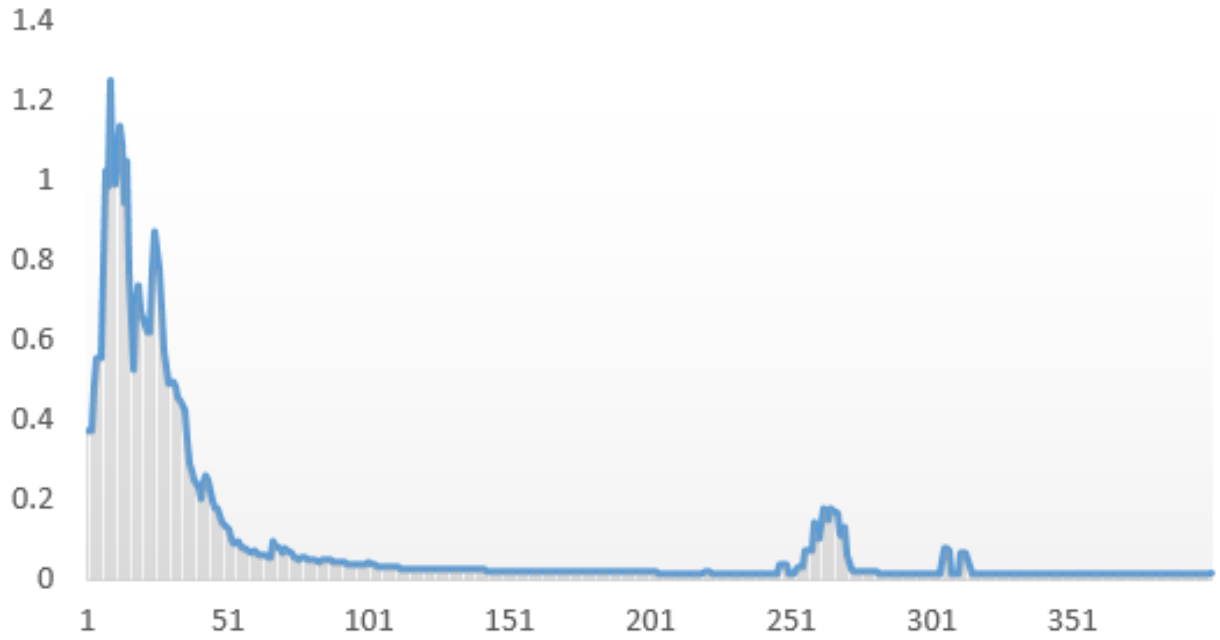


Рисунок 3.2 – Среднеквадратичная ошибка первых 400 эпох модели 40-79-79-1

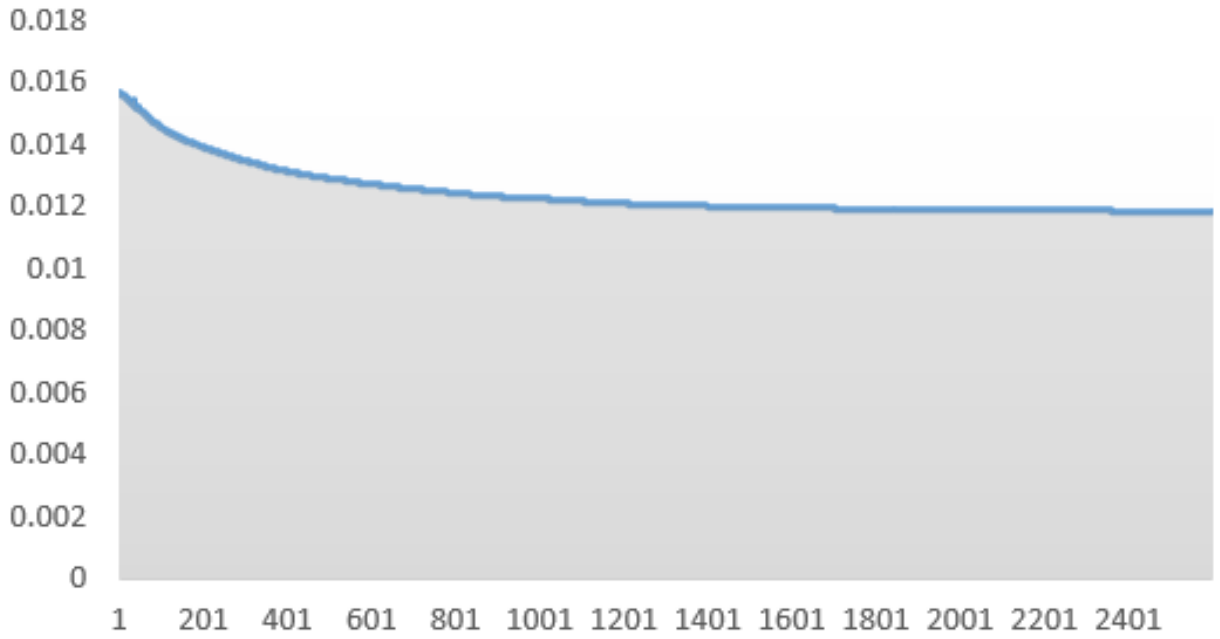
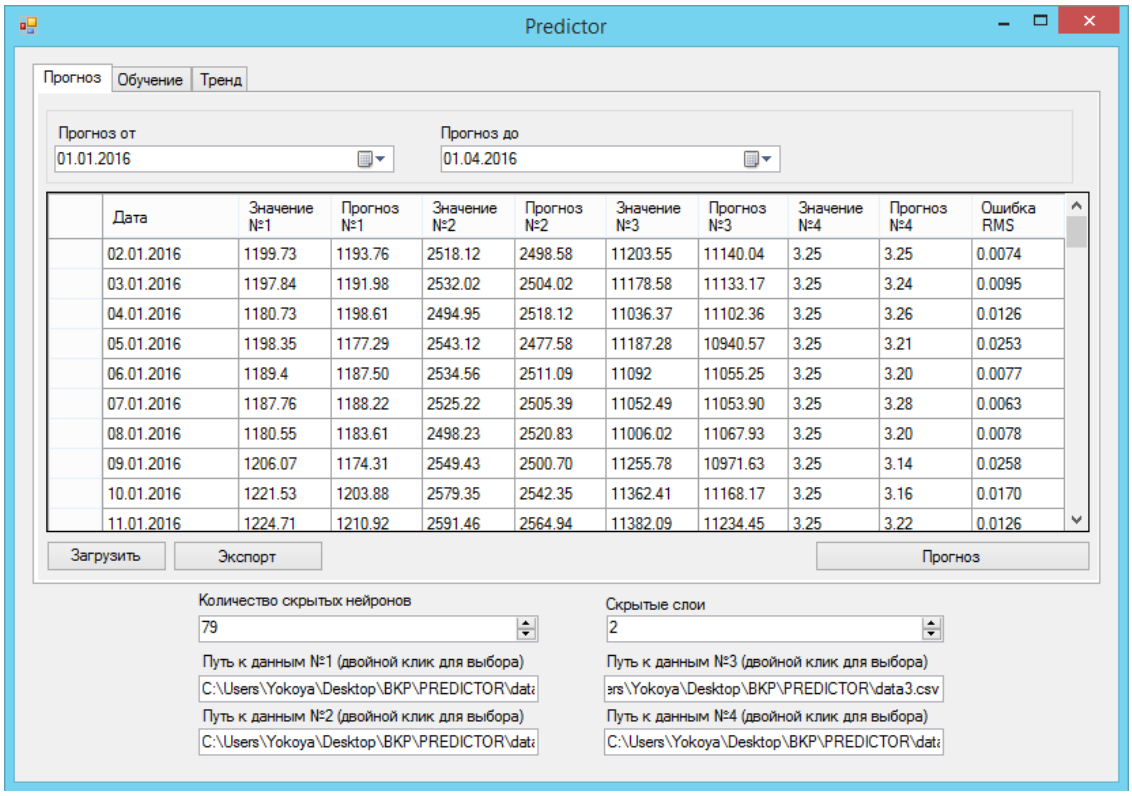


Рисунок 3.3 – Среднеквадратичная ошибка эпох с 400 по 3000 модели 40-79-79-4

3.2 Тестирование модуля «Прогноз»

В процессе обучения модели 40-79-79-4 на протяжении 3000 эпох, каждому из скрытых нейронов был присвоен свой вес. Используем результаты этого обучения, чтобы спрогнозировать значения для всех четырёх обучающих множеств на период с 1 января 2016 по 1 апреля 2016. Результаты работы модуля прогнозирования представлены на рисунке 3.4.



Дата	Значение №1	Прогноз №1	Значение №2	Прогноз №2	Значение №3	Прогноз №3	Значение №4	Прогноз №4	Ошибка RMS
02.01.2016	1199.73	1193.76	2518.12	2498.58	11203.55	11140.04	3.25	3.25	0.0074
03.01.2016	1197.84	1191.98	2532.02	2504.02	11178.58	11133.17	3.25	3.24	0.0095
04.01.2016	1180.73	1198.61	2494.95	2518.12	11036.37	11102.36	3.25	3.26	0.0126
05.01.2016	1198.35	1177.29	2543.12	2477.58	11187.28	10940.57	3.25	3.21	0.0253
06.01.2016	1189.4	1187.50	2534.56	2511.09	11092	11055.25	3.25	3.20	0.0077
07.01.2016	1187.76	1188.22	2525.22	2505.39	11052.49	11053.90	3.25	3.28	0.0063
08.01.2016	1180.55	1183.61	2498.23	2520.83	11006.02	11067.93	3.25	3.20	0.0078
09.01.2016	1206.07	1174.31	2549.43	2500.70	11255.78	10971.63	3.25	3.14	0.0258
10.01.2016	1221.53	1203.88	2579.35	2542.35	11362.41	11168.17	3.25	3.16	0.0170
11.01.2016	1224.71	1210.92	2591.46	2564.94	11382.09	11234.45	3.25	3.22	0.0126

Рисунок 3.4 – Прогнозные значения после 3000 эпох обучения модели 40-79-79-4

Как видно из рисунка 3.4, прогнозные значения значительно приближены к действительным значениям. Стоит отметить, что контрольные значения не участвовали в обучающей выборке, а значит данная ИНС проходит проверку на качество и может быть использована в процессах принятия решений.

Чем дальше горизонт прогнозирования от своей обучающей выборки, тем больше вырастает значение ошибки прогнозирования. Для получение более точных прогнозов необходимо переобучить ИНС учитывая полученные прогнозные значения или добавив значения контрольной выборки к значениям

обучающей выборки. Также для построения прогноза на последующие временные периоды необходимо изменить горизонт прогнозирования.

3.3 Тестирование модуля «Тренд»

Графическое представление временного ряда вместе с его трендом появляется после выбора файлов с пользовательскими данными и нажатия кнопки «Тренд». Временной ряд и трендовая компонента данных о посетителях кинотеатров представлены на рисунке 3.5.

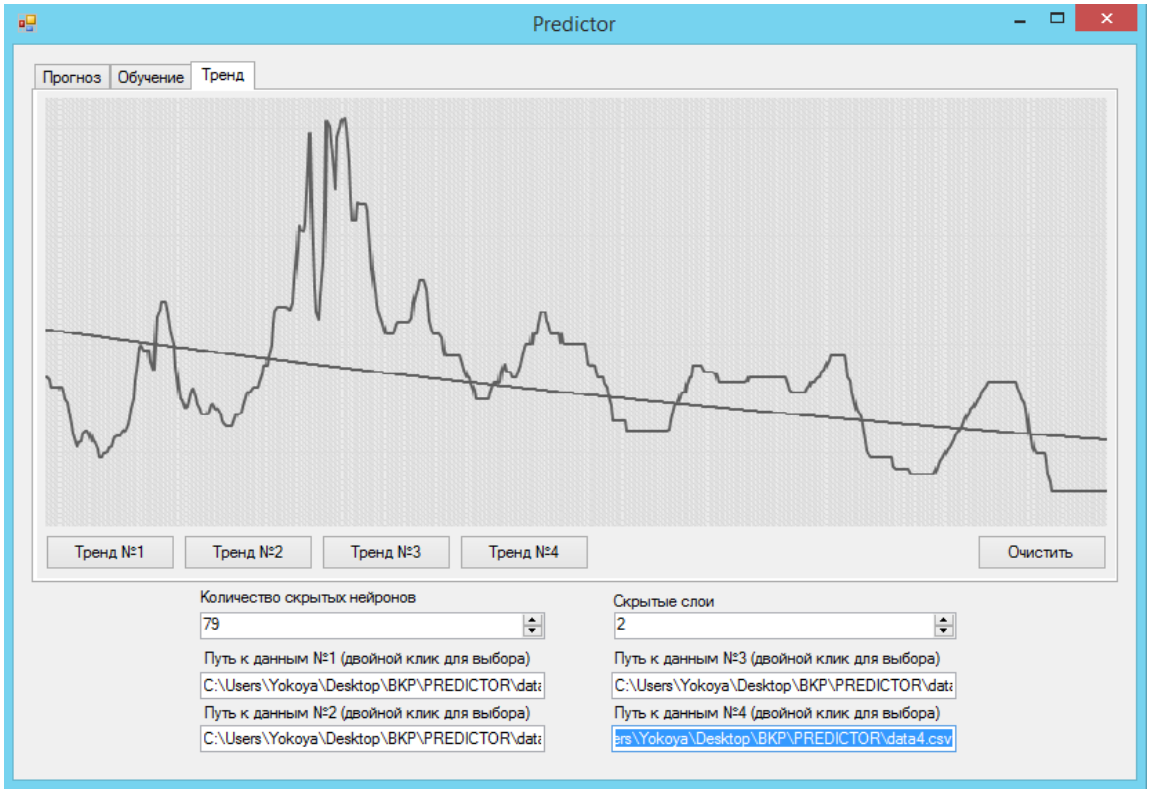


Рисунок 3.5 – Графическое представление и трендовая компонента временного ряда

Трендовая компонента выполняет свою функцию – показывает общую тенденцию развития данного временного ряда.

Как видно из рисунка 3.5, долговременная тенденция понижения посещаемости кинотеатра затемнена большим количеством колебаний. Таким образом, визуальный анализ графика не позволяет утверждать, что данные имеют тренд. В таких ситуациях и применяется МНК.

Заключение

Искусственные нейронные сети получили наибольшее распространение в области прогнозирования динамических систем, они успешно применяются для решения целого ряда задач интеллектуального анализа данных. Вместе с тем, для многих областей изучение возможностей применения нейронных сетей находится в экспериментальной стадии. Нейросетевые технологии не должны рассматриваться как универсальное средство решения всех подобных задач. Их применение оправдано в тех областях, в которых существует значительное число однотипных примеров, отражающих скрытые взаимосвязи.

В ходе бакалаврской работы были решены следующие задачи:

1. Рассмотрены нейросетевые методы анализа и прогнозирования.
2. Построена модель нейронной сети, соответствующая исходным данным
3. Спроектирован и разработан программный комплекс для анализа и прогнозирования временных рядов.
4. Получены результаты прогноза.
5. Оценена точность прогнозирования построенной модели.

Использование различных архитектур сетей, случайность выбора первоначальных синаптических коэффициентов и использование других сетевых параметров привели к тому, что прогнозы различных нейросетей, одинаково обученных на одних и тех же примерах отличались. Неудачные сети были отсеяны на этапе тестирования, а удачные использовались для принятия решений. Среднеквадратичная ошибка самой эффективной модели после 3000 эпох обучения составила 0.011838, что является хорошим результатом для моделей подобного рода. Исходя из полученных прогнозов и оценки построенной модели, можно сделать вывод, что искусственные нейронные сети являются хорошим инструментом для прогнозирования временных рядов, а

разработанный программный комплекс может быть использован в задачах связанных с принятием эффективных решений.

Таким образом, можно утверждать, что была достигнута цель и успешно решены все задачи бакалаврской работы.

Список используемой литературы

1. Подкорытова О. Анализ временных рядов: Учебное пособие / О.А. Подкорытова, М.В. Соколов – М.: Юрайт, 2016 – 266 с.

2. Садовникова Н. Анализ временных рядов и прогнозирование: Учебное пособие / Н.А. Садовникова, Р.А. Шмойлова – М.: Синергия, 2016 – 152 с.

3. Суслов В.И. Эконометрия. Часть III Эконометрия – I: Анализ временных рядов: Учебное пособие / В.И. Суслов, Н.М. Ибрагимов, Л.П. Талышева, А.А. Цыплаков – Новосибирск: СО РАН, 2005. – 744 с.

4. Файншмидт В. Дифференциальное и интегральное исчисление функций одного аргумента: Учебное пособие / В. Файншмидт – СПб.: БХВ-Петербург, 2006 – 224 с.

5. Ярушкина Н. Интеллектуальный анализ временных рядов: Учебное пособие / Н.Г. Ярушкина, Т.В. Афанасьева, И.Г. Перфильева – М.: Инфра-М, 2015 – 160 с.

6. Албахари Д. C# 5.0. Справочник. Полное описание языка / Д. Албахари, Б. Албахари – М.: Вильямс, 2013. – 1008 с.

7. Андерсон Т. Статистический анализ временных рядов / Т. Андерсон – М.: Вильямс, 2012. – 752 с.

8. Барсегян А. Технологии анализа данных. Data Mining, Visual Mining, Text Mining, OLAP / А. А. Барсегян, М. С. Куприянов, В. В. Степаненко, И. И. Холод – СПб.: БХВ-Петербург., 2007. – 384 с.

9. Васильев А. Принципы и техника нейросетевого моделирования / А.Н. Васильев, Д.А. Тархов – СПб.: Нестор-История, 2014 – 218 с.

10. Егорова Н. Прогнозирование фондовых рынков с использованием экономико-математических моделей / Н.Е. Егорова, А.Р Бахтизин Торжевский К.А. М.: Красанд, 2013 – 216 с.

11. Петцольд Ч. Программирование с использованием Microsoft Windows Forms / Ч. Петцольд – М.: Русская редакция, 2012 – 426 с.

12. Ширяев В. Принятие решений. Прогнозирование в глобальных системах / В.И Ширяев, Е.В Ширяев – М.: Либроком, 2013 – 216 с.

13. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен – М.: Вильямс, 2015. – 1312 с.

14. Хайкин С. Нейронные сети. Полный курс / С. Хайкин – М.: Вильямс, 2016. – 1104 с.

15. Анализ временных рядов [Электронный ресурс]: / Электрон. дан. - Идентификация модели временных рядов - http://www.statsoft.ru/home/textbook/modules/sttimser.html (дата обращения 26.04.2016).

16. Forecasting: principles and practice [Электронный ресурс]: / Электрон. дан. - Neural network models - https://www.otexts.org/fpp/9/3 (дата обращения 04.04.2016).

17. Heaton J. Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks / J. Heaton - CreateSpace Independent Publishing Platform, 2015 – 374 p.

18. Heaton J. Programming Neural Networks with Encog3 in C# - 2nd Edition / J. Heaton - Heaton Research, Incorporated, 2011 – 240 p.

19. Leondes T. Neural Network Systems Techniques and Applications Vol. 7 / C.T. Leondes – San Diego: Academic Press, 2010 – 438 p.

20. Montgomery D. Introduction to Time Series Analysis and Forecasting - 2nd Edition / D.C. Montgomery, C.L. Jennings, M. Kulahci – New York: Wiley, 2015 – 672 p.

Приложение А

Код класса NeuroPredictor

```
using System;
using System . Collections . Generic;
using System . Globalization;
using System . IO;
using System . Windows . Forms;
using NeuroPredictor . Entities;
using NeuroPredictor . Properties;
using NeuroPredictor . Utilities;
using System . Configuration;
using System . Security . Permissions;

namespace NeuroPredictor
{
    public partial class NeuroPredictor : Form
    {
        #region Private member fields
        private string _pathToData1 = "data1 . csv";
        private string _pathToData2 = "data2 . csv";
        private string _pathToData4 = "data3 . csv";
        private string _pathToData3 = "data4 . csv";
        private PredictIndicators _predictor;
        private readonly DateTime _predictFrom = CSVReader .  ParseDate ( "2016-01-01" ) ;
        private readonly DateTime _predictTo = CSVReader .  ParseDate ( "2016-04-01" ) ;
        private readonly DateTime _learnFrom = CSVReader .  ParseDate ( "2011-01-01" ) ;
        private readonly DateTime _learnTo = CSVReader .  ParseDate ( "2016-01-01" ) ;
        private readonly DateTime _maxDate;
        private readonly DateTime _minDate;
        private int _hidden Layers  = 2;
        private int _hiddenUnits = 41;
        private bool _reloadFiles = false;
        #endregion
        public NeuroPredictor (  )
        {
            InitializeComponent (  ) ;
            _btnStop . Enabled = false;
            _btnExport . Enabled = false;
            try
            {
                _maxDate = CSVReader .  ParseDate  ( ConfigurationManager .
AppSettings["MaxDate"] ) ;
                _minDate = CSVReader .  ParseDate  ( ConfigurationManager .
AppSettings["MinDate"] ) ;
            }
            catch
            {
                _maxDate = DateTime . Now;
                _minDate = CSVReader .  ParseDate ( "1971-02-05" ) ;
            }

            _dtpTrainFrom .  Value = _learnFrom;
            _dtpTrainUntil .  Value = _learnTo;
            _dtpPredictFrom .  Value = _predictFrom;
            _dtpPredictTo .  Value = _predictTo;

            _dtpTrainFrom .  MaxDate = _dtpTrainUntil .  MaxDate = _dtpPredictFrom .
MaxDate = _dtpPredictTo .  MaxDate = _maxDate;
```

```
            _dtpTrainFrom . MinDate = _dtpTrainUntil . MinDate = _dtpPredictFrom .
MinDate = _dtpPredictTo . MinDate = _minDate;

            _nudHidden Layers . Value = _hidden Layers ;
            _nudHiddenUnits . Value = _hiddenUnits;

        }

        private void NeuroPredictorLoad ( object sender, EventArgs e )
        {
            SetPathsIn TextBoxes ( ) ;
        }

        private void SetPathsInT extBoxes ( )
        {
            if ( File . Exists ( Path . GetFullPath ( _pathToData1 ) ) )
                _tbPathToData1 . Text = Path . GetFileName ( _pathToData1 ) ;
            if ( File . Exists ( Path . GetFullPath ( _pathToData2 ) ) )
                _tbPathToData2 . Text = Path . GetFileName ( _pathToData2 ) ;
            if ( File . Exists ( Path . GetFullPath ( _pathToData3 ) ) )
                _tbPathToData3 . Text = Path . GetFileName ( _pathToData3 ) ;
            if ( File . Exists ( Path . GetFullPath ( _pathToData4 ) ) )
                _tbPathToData4 . Text = Path . GetFileName ( _pathToData4 ) ;
        }

        private void TrainingCallback ( int epoch, double error, TrainingAlgorithm
algorithm )
        {
            Invoke ( addAction, new object [] {epoch, error, algorithm, _dgvTraining
Results} ) ;

        }

        private void BtnStart TrainingClick ( object sender, EventArgs e )
        {
            if ( _dgvTrainingResults . Rows . Count != 0 )
                _dgvTrainingResults . Rows . Clear ( ) ;

            if ( _predictor == null )
            {
                _reloadFiles = false;
                if ( !File . Exists ( _tbPathToData3 . Text ) || !File . Exists (
_tbPathToData4 . Text ) ||
                    !File . Exists ( _tbPathToData2 . Text ) || !File . Exists (
_tbPathToData1 . Text ) )
                {
                    MessageBox . Show ( Resources . InputMissing, Resources .
FileMissing, MessageBoxButtons . OK, MessageBoxIcon . Information ) ;
                    return;
                }
            }

            DateTime trainFrom = _dtpTrainFrom . Value;
            DateTime trainTo = _dtpTrainUntil . Value;

            if ( trainFrom > trainTo )
            {
                MessageBox . Show ( Resources . TrainFrom TrainTo, Resources .
BadParameters, MessageBoxButtons . OK, MessageBoxIcon . Information ) ;
                _dtpTrainFrom . Focus ( ) ;
                return;
            }
            Fade Controls ( true ) ;
            if ( _predictor == null )
```

```
            {
                _pathToData1 = _tbPathToData1 . Text;
                _pathToData3 = _tbPathToData3 . Text;
                _pathToData4 = _tbPathToData4 . Text;
                _pathToData2 = _tbPathToData2 . Text;
                 Cursor = Cursors .  WaitCursor;
                _hidden Layers  =  ( int ) _nudHidden Layers  .  Value;
                _hiddenUnits =  ( int ) _nudHiddenUnits .  Value;
                try
                {
                    _predictor = new Predict Indicators  ( _pathToData1, _pathToData2,
_pathToData3, _pathToData4, _hiddenUnits, _hidden Layers  ) ;
                }
                catch  ( Exception ex )
                {
                    MessageBox .  Show  ( ex . Message, Resources .  Exception,
MessageBoxButtons .  OK, MessageBoxIcon .  Error ) ;
                    _predictor = null;
                    return;
                }
                finally
                {
                    Cursor = Cursors .  Default;
                }
            }
            else if  ( _reloadFiles )
            {
                _pathToData1 = _tbPathToData1 . Text;
                _pathToData3 = _tbPathToData3 . Text;
                _pathToData4 = _tbPathToData4 . Text;
                _pathToData2 = _tbPathToData2 . Text;
                _predictor .  ReloadFiles ( _pathToData1, _pathToData2, _pathToData3,
_pathToData4 ) ;
                _dtpTrainFrom .  MinDate = _predictor .  MinIndexDate;
                _dtpTrainUntil .  MaxDate = _predictor .  MaxIndexDate;
            }

            if  ( trainFrom < _predictor .  MinIndexDate )
                _dtpTrain From .  MinDate = _dtpTrain From .  Value = trainFrom = _predictor
.  MinIndexDate;
            if  ( trainTo > _predictor .  MaxIndexDate )
                _dtpTrain Until .  MaxDate = _dtpTrain Until .  Value = trainTo = _predictor
.  MaxIndexDate;
            TrainingStatus callback = Training Callback;
            _predictor .  Train Network Async  ( trainFrom, trainTo, callback ) ;
        }

        private void BtnPredict Click  ( object sender, EventArgs e )
        {
            if  ( _dgv Prediction Results .  Rows . Count != 0 )
                _dgv Prediction Results .  Rows . Clear (  ) ;

            if  ( _predictor == null )
            {
                _reloadFiles = false;
                if  ( !File . Exists  ( _tbPathToData3 . Text )  || !File . Exists  (
_tbPathToData4 . Text )  ||
                    !File . Exists  ( _tbPathToData2 . Text )  || !File . Exists  (
_tbPathToData1 . Text )  )
                {
                    MessageBox .  Show  ( Resources .  InputMissing , Resources .
FileMissing, MessageBoxButtons .  OK, MessageBoxIcon .  Information ) ;
                    return;
                }
```

```
                switch  ( MessageBox .  Show  ( Resources .  Untrained Predictor Warning,
Resources .  NoNetwork, Message Box Buttons .  YesNoCancel, MessageBoxIcon .  Information )
)
                {
                    case DialogResult . Yes:
                        break;
                    case DialogResult . No:
                        this . Cursor = Cursors . WaitCursor;
                        _hidden Layers  =  ( int ) _nudHidden Layers  . Value;
                        _hiddenUnits =  ( int ) _nudHiddenUnits . Value;
                        try
                        {
                            _predictor = new PredictIndicators ( _pathToData1, _pathToData2,
_pathToData3, _pathToData4, _hiddenUnits, _hidden Layers  ) ;
                        }
                        catch  ( Exception ex )
                        {
                            MessageBox . Show ( ex . Message, Resources . Exception,
MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                            _predictor = null;
                            return;
                        }
                        finally
                        {
                            this . Cursor = Cursors . Default;
                        }
                        using  ( OpenFileDialog ofd = new OpenFileDialog (  )  { FileName =
"predictor . ntwrk", Filter = Resources . NtwrkFilter } )
                        {
                            if  ( ofd . ShowDialog (  )  == System .  Windows . Forms .
DialogResult . OK )
                            {
                                try
                                {
                                    _predictor . LoadNeuralNetwork ( Path . GetFullPath (
ofd . FileName )  ) ;
                                }
                                catch
                                {
                                    MessageBox . Show ( Resources . ExceptionMessage,
Resources . Exception, MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                                    return;
                                }
                            }
                        }
                        break;
                    case DialogResult . Cancel:
                        return;
                }
            }
            DateTime predictFrom = _dtpPredictFrom . Value;
            DateTime predictTo = _dtpPredictTo . Value;
            if  ( predictFrom > predictTo )
            {
                MessageBox . Show ( Resources . PredictFromToWarning, Resources .
BadParameters, MessageBoxButtons . OK, MessageBoxIcon . Information ) ;
                _dtpPredictFrom . Focus (  ) ;
                return;
            }

            if  ( _predictor == null )
            {
                _pathToData1 = _tbPathToData1 . Text;
                _pathToData3 = _tbPathToData3 . Text;
```

```
                _pathToData4 = _tbPathToData4 . Text;
                _pathToData2 = _tbPathToData2 . Text;
                 Cursor = Cursors . WaitCursor;
                _hidden Layers  =  ( int ) _nudHidden Layers  . Value;
                _hiddenUnits =  ( int ) _nudHiddenUnits . Value;
                try
                {
                    _predictor = new PredictIndicators ( _pathToData1, _pathToData2,
_pathToData3, _pathToData4, _hiddenUnits, _hidden Layers  ) ;
                }
                catch  ( Exception ex )
                {
                    MessageBox . Show ( ex . Message, Resources . Exception,
MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                    _predictor = null;
                    return;
                }
                finally
                {
                    Cursor = Cursors . Default;
                }
            }
            List<PredictionResults> results = null;
            try
            {
                if  ( _reloadFiles )
                {
                    _pathToData1 = _tbPathToData1 . Text;
                    _pathToData3 = _tbPathToData3 . Text;
                    _pathToData4 = _tbPathToData4 . Text;
                    _pathToData2 = _tbPathToData2 . Text;
                    _predictor . ReloadFiles ( _pathToData1, _pathToData2, _pathToData3,
_pathToData4 ) ;
                }
                results = _predictor . Predict ( predictFrom, predictTo ) ;
            }
            catch ( Exception ex )
            {
                MessageBox . Show ( ex . Message, Resources . Exception, MessageBoxButtons .
OK, MessageBoxIcon . Error ) ;
                return;
            }
            foreach  ( var item in results )
            {
                _dgvPredictionResults . Rows . Add ( item . Date . ToShortDateString (  ) ,
item . ActualOne,
                                                    item . PredictedOne . ToString ( "F2",
CultureInfo . InvariantCulture ) , item . ActualFour, item . PredictedFour . ToString (
"F2", CultureInfo . InvariantCulture ) , item . ActualThree,
                                                    item . PredictedThree . ToString ( "F2",
CultureInfo . InvariantCulture ) , item . ActualTwo, item . PredictedTwo . ToString ( "F2",
CultureInfo . InvariantCulture ) , item . Error . ToString ( "F4", CultureInfo .
InvariantCulture )  ) ;
            }
        }

        private void NeuroPredictorFormClosing ( object sender, FormClosingEventArgs e )
        {
            if ( _predictor != null )
                _predictor . AbortTraining (  ) ;
        }

        private void BtnStopClick ( object sender, EventArgs e )
        {
```

```
            FadeControls ( false ) ;
            _predictor . AbortTraining (  ) ;
            _btnExport . Enabled = true;
        }

        private void TbPathToData1MouseDoubleClick ( object sender, MouseEventArgs e )
        {
            OpenFileDialog ofd = new OpenFileDialog (  )  { FileName = "data1 . csv", Filter
= Resources . CsvFilter };
            if  ( ofd . ShowDialog (  )  == System .  Windows . Forms . DialogResult . OK )
            {
                _tbPathToData1 . Text = Path . GetFullPath ( ofd . FileName ) ;
                _reloadFiles = true;
            }
        }

        private void TbPathToData2MouseDoubleClick ( object sender, MouseEventArgs e )
        {
            OpenFileDialog ofd = new OpenFileDialog (  )  { FileName = "data2 . csv", Filter
= Resources . CsvFilter };
            if  ( ofd . ShowDialog (  )  == System .  Windows . Forms . DialogResult . OK )
            {
                _tbPathToData2 . Text = Path . GetFullPath ( ofd . FileName ) ;
                _reloadFiles = true;
            }
        }

        private void TbPathToData3MouseDoubleClick ( object sender, MouseEventArgs e )
        {
            OpenFileDialog ofd = new OpenFileDialog (  )  { FileName = "data3 . csv", Filter
= Resources . CsvFilter };
            if  ( ofd . ShowDialog (  )  == System .  Windows . Forms . DialogResult . OK )
            {
                _tbPathToData3 . Text = Path . GetFullPath ( ofd . FileName ) ;
                _reloadFiles = true;
            }
        }

        private void TbPathToData4MouseDoubleClick ( object sender, MouseEventArgs e )
        {
            OpenFileDialog ofd = new OpenFileDialog (  )  { FileName = "data4 . csv", Filter
= Resources . CsvFilter };
            if  ( ofd . ShowDialog (  )  == System .  Windows . Forms . DialogResult . OK )
            {
                _tbPathToData4 . Text = Path . GetFullPath ( ofd . FileName ) ;
                _reloadFiles = true;
            }
        }

        private void FadeControls ( bool fade )
        {
            Action<bool> action =  ( param )  =>
                                   {
                                       _tbPathToData1 . Enabled = param;
                                       _tbPathToData2 . Enabled = param;
                                       _tbPathToData3 . Enabled = param;
                                       _tbPathToData4 . Enabled = param;
                                       _btnStartTraining . Enabled = param;
                                       _btnStop . Enabled = !param;
                                       _dtpPredictFrom . Enabled = param;
                                       _dtpPredictTo . Enabled = param;
                                       _dtpTrainFrom . Enabled = param;
                                       _dtpTrainUntil . Enabled = param;
                                       _nudHidden Layers  . Enabled = param;
```

```
                                    _nudHiddenUnits . Enabled = param;
                                };
            Invoke ( action, !fade ) ;
        }

        private void BtnExportClick ( object sender, EventArgs e )
        {
            using ( SaveFileDialog sfd = new SaveFileDialog (  )  { FileName = "predictor .
ntwrk", Filter = Resources . NtwrkFilter } )
            {
                if  ( sfd . ShowDialog (  )  == DialogResult . OK )
                {
                    FileIOPermission perm = new FileIOPermission ( FileIOPermissionAccess .
Write, Path . GetFullPath ( sfd . FileName )  ) ;
                    try
                    {
                        perm . Demand (  ) ;
                    }
                    catch  ( System .  Security . SecurityException )
                    {
                        MessageBox . Show ( Resources . SecurityExceptionMessage, Resources
. SecurityException, MessageBoxButtons . OK, MessageBoxIcon . Exclamation ) ;
                        return;
                    }
                    _predictor . ExportNeuralNetwork ( Path . GetFullPath ( sfd . FileName )
) ;
                }
            }
        }

        private void BtnLoadClick ( object sender, EventArgs e )
        {
            if  ( !File . Exists ( _tbPathToData3 . Text )  || !File . Exists (
_tbPathToData4 . Text )  ||
                !File . Exists ( _tbPathToData2 . Text )  || !File . Exists (
_tbPathToData1 . Text )  )
            {
                MessageBox . Show ( Resources . InputMissing, Resources . FileMissing,
MessageBoxButtons . OK, MessageBoxIcon . Information ) ;
                return;
            }
            if  ( _predictor == null || _predictor . Loaded == false )
            {
                this . Cursor = Cursors . WaitCursor;
                _hidden Layers  =  ( int ) _nudHidden Layers  . Value;
                _hiddenUnits =  ( int ) _nudHiddenUnits . Value;
                try
                {
                    _predictor = new PredictIndicators ( _pathToData1, _pathToData2,
_pathToData3, _pathToData4, _hiddenUnits, _hidden Layers  ) ;
                }
                catch  ( Exception ex )
                {
                    MessageBox . Show ( ex . Message, Resources . Exception,
MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                    _predictor = null;
                    return;
                }
                finally
                {
                    this . Cursor = Cursors . Default;
                }
            }
```

```
            using  ( OpenFileDialog ofd = new OpenFileDialog (  )  { FileName = "predictor .
ntwrk", Filter = Resources . NtwrkFilter } )
            {
                if  ( ofd . ShowDialog (  )  == DialogResult . OK )
                {
                    try
                    {
                        _predictor . LoadNeuralNetwork ( Path . GetFullPath ( ofd . FileName
) ) ;
                        _nudHidden Layers  . Value = _predictor . Hidden Layers ;
                        _nudHiddenUnits . Value = _predictor . HiddenUnits;
                    }
                    catch  ( System .  Security . SecurityException )
                    {
                        MessageBox . Show ( Resources . SecurityExceptionFolderLevel,
Resources . Exception, MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                    }
                    catch
                    {
                        MessageBox . Show ( Resources . ExceptionMessage, Resources .
Exception, MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                        return;
                    }
                }
            }
        }

        private void BtnSaveResultsClick ( object sender, EventArgs e )
        {
            var dgvResults = _dgvPredictionResults;
            SaveFileDialog ofd = new SaveFileDialog {Filter = Resources . CsvFilter,
FileName = "results . csv"};
            if  ( ofd . ShowDialog (  )  == DialogResult . OK )
            {
                CSVWriter writer = null;
                try
                {
                    writer = new CSVWriter ( ofd . FileName ) ;
                }
                catch  ( System .  Security . SecurityException )
                {
                    MessageBox . Show (  Resources . SecurityExceptionFolderLevel, Resources
. Exception, MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                    return;
                }
                catch ( Exception ex )
                {
                    MessageBox . Show ( ex . Message, Resources . Exception,
MessageBoxButtons . OK, MessageBoxIcon . Error ) ;
                    return;
                }
                object[,] values = new object[dgvResults . Rows . Count + 2,dgvResults .
Columns . Count];
                int rowIndex = 0;
                int colIndex = 0;
                foreach  ( DataGridViewColumn col in dgvResults . Columns )
                {
                    values[rowIndex, colIndex] = col . HeaderText;
                    colIndex++;
                }
                rowIndex++;

                foreach  ( DataGridViewRow row in dgvResults . Rows )
                {
```

```
                    colIndex = 0;
                    foreach  ( DataGridViewCell cell in row . Cells )
                    {
                        values[rowIndex, colIndex] = cell . Value;
                        colIndex++;
                    }
                    rowIndex++;
                }
                writer . Write ( values ) ;
            }
        }
        private void NudHiddenUnitsValueChanged ( object sender, EventArgs e )
        {
            if ( _predictor != null )
            {
                if ( MessageBox . Show ( Resources . ChangedNetwork, Resources . Warning,
MessageBoxButtons . OKCancel, MessageBoxIcon . Warning )  == DialogResult . OK )
                {
                    _predictor = null;
                }
            }
        }
        private void NudHidden Layers ValueChanged ( object sender, EventArgs e )
        {
            if  ( _predictor != null )
            {
                if  ( MessageBox . Show ( Resources . ChangedNetwork, Resources . Warning,
MessageBoxButtons . OKCancel, MessageBoxIcon . Warning )  == DialogResult . OK )
                {
                    _predictor = null;
                }
            }
        }

        private void _labPathToSp_Click ( object sender, EventArgs e )
        {

        }

        private void _gbPredict_Enter ( object sender, EventArgs e )
        {

        }

        private void button1_Click ( object sender, EventArgs e )
        {

        }

        private void _tbPathToSp_TextChanged ( object sender, EventArgs e )
        {

        }

        private void _dgvPredictionResults_CellContentClick ( object sender,
DataGridViewCellEventArgs e )
        {

        }

        private void tabPage1_Click ( object sender, EventArgs e )
        {

        }
```

```
    }
}
```

Приложение Б

Программный код класса PredictIndicators

```
using System;
using System . Text;
using System . Collections . Generic;
using System . IO;
using System . Threading;
using NeuroPredictor . Entities;
using NeuroPredictor . Utilities;
using Encog . Neural . Networks;
using Encog . Neural . Networks . Layers;
using Encog . Neural . Networks . Training;
using Encog . Neural . Data . Basic;
using Encog . Neural . Networks . Training . LMA;
using Encog . Neural . Networks . Training . Propagation . Resilient;
using Encog . Neural . Networks . Training . Anneal;
using Encog . Neural . Networks . Training . Strategy;
using Encog . Neural . Activation;
using Encog . Persist . Persistors;

namespace NeuroPredictor
{
    public enum TrainingAlgorithm
    {
        Resilient,
        Annealing,
        Evolutionary
    }

    public delegate void TrainingStatus ( int iteration, double error, TrainingAlgorithm
algorithm ) ;
    public sealed class PredictIndicators
    {
        #region Constants
        private const int INDEXES_TO_CONSIDER = 4;
        private const int INPUT_TUPLES = 10;
        private const int OUTPUT_SIZE = 4;
        private const double MAX_ERROR = 0 . 00005;
        #endregion

        #region Private Members
        private BasicNetwork _network;
        private double[][] _input;
        private double[][] _ideal;
        private NeuroPredictorManager _manager;
        private Thread _trainThread;
        private string _pathtoone;
        private string _pathtotwo;
        private string _pathtothree;
        private string _pathtofour;
        private int _trainingSize = 1000;
        #endregion

        public bool Loaded { get; private set; }
        public int Hidden Layers  { get; private set; }
        public int HiddenUnits { get; private set; }
        public DateTime MaxIndexDate
```

```
        {
            get
            {
                return _manager == null ? DateTime . MinValue : _manager . MaxDate;
            }
        }

        public DateTime MinIndexDate
        {
            get
            {
                return _manager == null ? DateTime . MaxValue : _manager . MinDate;
            }
        }
        #region Constructors
        public PredictIndicators ( string pathToOne, string pathToTwo, string pathToThree,
string pathToFour, int hiddenUnits, int hidden Layers  )
        {
            if  ( !File . Exists ( pathToOne )  )
                throw new ArgumentException ( "Неправильный путь к данным №1" ) ;
            if  ( !File . Exists ( pathToTwo )  )
                throw new ArgumentException ( "Неправильный путь к данным №2" ) ;
            if  ( !File . Exists ( pathToThree )  )
                throw new ArgumentException ( "Неправильный путь к данным №3" ) ;
            if  ( !File . Exists ( pathToFour )  )
                throw new ArgumentException ( "Неправильный путь к данным №4" ) ;

            _pathtoone = pathToOne;
            _pathtotwo = pathToTwo;
            _pathtothree = pathToThree;
            _pathtofour = pathToFour;

            CreateNetwork ( hiddenUnits, hidden Layers  ) ;
            _manager = new NeuroPredictorManager ( INPUT_TUPLES, OUTPUT_SIZE ) ;
            _manager . Load ( _pathtoone, _pathtotwo, _pathtothree, _pathtofour ) ;
            Loaded = true;
            Hidden Layers  = hidden Layers ;
            HiddenUnits = hiddenUnits;
        }
        #endregion

        public void ReloadFiles ( string pathToOne, string pathToTwo, string pathToThree,
string pathTofour )
        {
            if  ( !File . Exists ( pathToOne )  )
                throw new ArgumentException ( "Неправильный путь к данным №1" ) ;
            if  ( !File . Exists ( pathToTwo )  )
                throw new ArgumentException ( "Неправильный путь к данным №2" ) ;
            if  ( !File . Exists ( pathToThree )  )
                throw new ArgumentException ( "Неправильный путь к данным №3" ) ;
            if  ( !File . Exists ( pathTofour )  )
                throw new ArgumentException ( "Неправильный путь к данным №4" ) ;
            Loaded = false;
            _pathtoone = pathToOne;
            _pathtotwo = pathToTwo;
            _pathtothree = pathToThree;
            _pathtofour = pathTofour;
            _manager = new NeuroPredictorManager ( INPUT_TUPLES, OUTPUT_SIZE ) ;
            _manager . Load ( _pathtoone, _pathtotwo, _pathtothree, _pathtofour ) ;
            _ideal = _input = null;
            Loaded = true;
        }

        private void CreateNetwork ( int hiddenUnits, int hidden Layers  )
```

```
        {
            _network = new BasicNetwork {Name = "Predictor", Description = "Network for
prediction analysis"};
            _network . AddLayer ( new BasicLayer ( INPUT_TUPLES * INDEXES_TO_CONSIDER )  ) ;
            for  ( int i = 0; i < hidden Layers ; i++ )
                _network . AddLayer ( new BasicLayer ( new ActivationTANH (  ) , true,
hiddenUnits )  ) ;
            _network . AddLayer ( new BasicLayer ( new ActivationTANH (  ) , true,
OUTPUT_SIZE )  ) ;
            _network . Structure . FinalizeStructure (  ) ;
            _network . Reset (  ) ;
        }

        public void CreateTrainingSets ( DateTime trainFrom, DateTime trainTo )
        {
            int startIndex = -1;
            int endIndex = -1;
            foreach  ( UserData sample in _manager . Samples )
            {
                if  ( sample . Date . CompareTo ( trainFrom )  < 0 )
                    startIndex++;
                if  ( sample . Date . CompareTo ( trainTo )  < 0 )
                    endIndex++;
            }
            _trainingSize = endIndex - startIndex;
            _input = new double[_trainingSize][];
            _ideal = new double[_trainingSize][];

            for  ( int i = startIndex; i < endIndex; i++ )
            {
                _input[i - startIndex] = new double[INPUT_TUPLES * INDEXES_TO_CONSIDER];
                _ideal[i - startIndex] = new double[OUTPUT_SIZE];
                _manager . GetInputData ( i, _input[i - startIndex] ) ;
                _manager . GetOutputData ( i, _ideal[i - startIndex] ) ;
            }
        }

        public void TrainNetworkAsync ( DateTime trainFrom, DateTime trainTo, TrainingStatus
status )
        {
            Action<DateTime, DateTime, TrainingStatus> action = TrainNetwork;
            action . BeginInvoke ( trainFrom, trainTo, status, action . EndInvoke, action )
;
        }

        private void TrainNetwork ( DateTime trainFrom, DateTime trainTo, TrainingStatus
status )
        {
            if ( _input == null || _ideal == null )
                CreateTrainingSets ( trainFrom, trainTo ) ;
            _trainThread = Thread . CurrentThread;
            int epoch = 1;
            ITrain train = null;
            try
            {
                var trainSet = new BasicNeuralDataSet ( _input, _ideal ) ;
                train = new ResilientPropagation ( _network, trainSet ) ;
                double error;
                do
                {
                    train . Iteration (  ) ;
                    error = train . Error;
                    if  ( status != null )
                        status . Invoke ( epoch, error, TrainingAlgorithm . Resilient ) ;
```

```
                    epoch++;
                } while  ( error > MAX_ERROR ) ;
            }
            catch  ( ThreadAbortException )  { _trainThread = null; }
            finally
            {
                train . FinishTraining (  ) ;
            }
            _trainThread = null;
        }

        public void AbortTraining (  )
        {
            if  ( _trainThread != null )  _trainThread . Abort (  ) ;
        }

        [System .  Security . Permissions . FileIOPermission ( System .  Security .
Permissions . SecurityAction . Demand ) ]
        public void ExportNeuralNetwork ( string path )
        {
            if  ( _network == null )
                throw new NullReferenceException ( "Сеть заполнена нулевыми значениями .
Экспорт невозможен . " ) ;
            Encog . Util . SerializeObject . Save ( path, _network ) ;
        }

        public void LoadNeuralNetwork ( string path )
        {
            _network =  ( BasicNetwork ) Encog . Util . SerializeObject . Load ( path ) ;
            Hidden Layers  = _network . Structure . Layers . Count - 2;
            HiddenUnits = _network . Structure . Layers[1] . NeuronCount;
        }

        public List<PredictionResults> Predict ( DateTime predictFrom, DateTime predictTo )
        {
            List<PredictionResults> results = new List<PredictionResults> (  ) ;
            double[] present = new double[INPUT_TUPLES * INDEXES_TO_CONSIDER];
            double[] actualOutput = new double[OUTPUT_SIZE];
            int index = 0;
            foreach  ( var sample in _manager . Samples )
            {
                if  ( sample . Date . CompareTo ( predictFrom )  > 0 && sample . Date .
CompareTo ( predictTo )  < 0 )
                {
                    PredictionResults result = new PredictionResults (  ) ;
                    _manager . GetInputData ( index - INPUT_TUPLES, present ) ;
                    _manager . GetOutputData ( index - INPUT_TUPLES, actualOutput ) ;
                    var data = new BasicNeuralData ( present ) ;
                    var predict = _network . Compute ( data ) ;
                    result . ActualOne = actualOutput[0] *  ( _manager . MaxOne - _manager .
MinOne )  + _manager . MinOne;
                    result . PredictedOne = predict[0] *  ( _manager . MaxOne - _manager .
MinOne )  + _manager . MinOne;
                    result . ActualTwo = actualOutput[1] *  ( _manager . MaxTwo - _manager .
MinTwo )  + _manager . MinTwo;
                    result . PredictedTwo = predict[1] *  ( _manager . MaxTwo - _manager .
MinTwo )  + _manager . MinTwo;
                    result . ActualThree = actualOutput[2] *  ( _manager . MaxThree -
_manager . MinThree )  + _manager . MinThree;
                    result . PredictedThree = predict[2] *  ( _manager . MaxThree - _manager
. MinThree )  + _manager . MinThree;
                    result . ActualFour = actualOutput[3] *  ( _manager . MaxFour - _manager
. MinFour )  + _manager . MinFour;
```

```
                    result . PredictedFour = predict[3] *  ( _manager . MaxFour - _manager .
MinFour )  + _manager . MinFour;
                    result . Date = sample . Date;
                    ErrorCalculation error = new ErrorCalculation (  ) ;
                    error . UpdateError ( actualOutput, predict . Data ) ;
                    result . Error = error . CalculateRMS (  ) ;
                    results . Add ( result ) ;
                }
                index++;
            }
            return results;
        }
    }
}
```

Приложение В

Программный код класса CSVReader

```
using System;
using System . Collections . Generic;
using System . Globalization;
using System . IO;

namespace NeuroPredictor . Utilities
{
    public class CSVReader : IDisposable
    {
        private readonly TextReader _reader;
        private readonly IDictionary<String, int> _columns = new Dictionary<String, int> (
) ;
        private readonly String[] _data;
        public static String DisplayDate ( DateTime date )
        {
            return date . ToString ( ) ;
        }
        public static DateTime ParseDate ( String when )
        {
            try
            {
                return DateTime . ParseExact ( when, "yyyy-MM-dd", CultureInfo .
InvariantCulture ) ;
            }
            catch ( FormatException )
            {
                return default ( DateTime ) ;
            }
        }
        public CSVReader ( String filename )
        {
            _reader = new StreamReader ( filename ) ;
            String line = _reader . ReadLine ( ) ;
            string[] tok = line . Split ( ',' ) ;

            for ( int index = 0; index < tok . Length; index++ )
            {
                String header = tok[index];
                _columns . Add ( header . ToLower ( ) , index ) ;
            }

            _data = new String[tok . Length];
        }
        public void Close ( )
        {
            _reader . Close ( ) ;
        }

        public String Get ( int i )
        {
            return _data[i];
        }

        public String Get ( String column )
        {
            if ( !_columns . ContainsKey ( column . ToLower ( ) ) )
            {
```

```
            return null;
        }
        int i = _columns[column . ToLower (  ) ];

        return _data[i];
    }
    public DateTime GetDate ( String column )
    {
        String str = Get ( column ) ;
        return DateTime . Parse ( str, CultureInfo . InvariantCulture ) ;
    }
    public double GetDouble ( String column )
    {
        String str = Get ( column ) ;
        return double . Parse ( str, CultureInfo . InvariantCulture ) ;
    }
    public int GetInt ( String col )
    {
        String str = Get ( col ) ;
        try
        {
            return int . Parse ( str, CultureInfo . InvariantCulture ) ;
        }
        catch  ( FormatException )
        {
            return 0;
        }
    }
    public bool Next (  )
    {
        String line = _reader . ReadLine (  ) ;
        if  ( line == null )
        {
            return false;
        }

        string[] tok = line . Split ( ',' ) ;

        for  ( int i = 0; i < tok . Length; i++ )
        {
            String str = tok[i];
            if  ( i < _data . Length )
            {
                _data[i] = str;
            }
        }

        return true;
    }

    #region IDisposable Members
    private bool _alreadydisposed = false;
    public void Dispose (  )
    {
        Dispose ( true ) ;
        _alreadydisposed = true;
        GC . SuppressFinalize ( this ) ;
    }

    protected virtual void Dispose ( bool isDisposing )
    {
        if ( !_alreadydisposed )
        {
            if ( isDisposing )
```

```
                {
                    _reader . Dispose (  ) ;
                }
            }
        }

        ~CSVReader (  )
        {
            Dispose ( false ) ;
        }
        #endregion
    }
}
```

Приложение Г

Программный код класса ErrorCalculation

```
using System;

namespace NeuroPredictor .  Utilities
{
    public class ErrorCalculation
    {
        private double _globalError;
        private int _setSize;
        public double CalculateRMS (  )
        {
            return Math . Sqrt ( _globalError/ ( _setSize )  ) ;
        }

        public void Reset (  )
        {
            _globalError = _setSize = 0;
        }

        public void UpdateError ( double[] actual, double[] ideal )
        {

            for  ( int i = 0; i < actual . Length; i++ )
            {
                double delta = ideal[i] - actual[i];
                _globalError += delta * delta;
            }
            _setSize += ideal . Length;
        }


    }
}
```