Пермский филиал федерального государственного автономного образовательного учреждения высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Образовательная программа магистратуры «Информационная аналитика в управлении предприятием»

**ОТЧЕТ**

**по проектной работе**

Выполнила студентка ИАУП-18-1

Неганова Эльвира Анатольевна

Руководитель проекта

доцент кафедры информационных технологий в бизнесе, к.т.н., доцент

Викентьева Ольга Леонидовна

\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

(дата)

Пермь, 2019

**Оглавление**

[Введение 3](#_Toc12445177)

[Глава 1. Выбор способа построения и типа нейронной сети 5](#_Toc12445178)

[1.1. Постановка задачи анализа данных 5](#_Toc12445179)

[1.2. Обоснование выбора языка Python для построения нейронной сети 6](#_Toc12445180)

[1.3. Обоснование выбора библиотеки Keras для построения нейронной сети 9](#_Toc12445181)

[1.4. Обоснование выбора архитектуры нейронной сети 11](#_Toc12445182)

[Глава 2. Предварительная обработка данных для нейронной сети 15](#_Toc12445183)

[2.1. Описание данных с датчиков, установленных в офисном помещении 15](#_Toc12445184)

[2.2. Подготовка данных для обучения нейронной сети 17](#_Toc12445185)

[Глава 3. Построение и обучение нейронной сети для обработки сигналов 20](#_Toc12445186)

[3.1. Построение нейронной сети 20](#_Toc12445187)

[3.2. Обучение нейронной сети 21](#_Toc12445188)

[3.3. Оценка качества работы нейросети 22](#_Toc12445189)

[Заключение 26](#_Toc12445190)

[Список литературы 27](#_Toc12445191)

[Приложение 28](#_Toc12445192)

[Приложение А. Заявка на проект 28](#_Toc12445193)

[Приложение Б. Программа нейронной сети 30](#_Toc12445194)

[Приложение B. Техническое задание 37](#_Toc12445195)

# Введение

Внедрение технологий Интернета вещей (IoT) позволяет улучшать производственные процессы, повышать энергоэффективность и создавать комфортные условия для работы и жизни людей путем объединения различных исполнительных механизмов, устройств и датчиков. Такие технологии делают возможным реализацию, например, интеллектуального отопления или интеллектуального учета электроэнергии. Датчики имеют решающее значение в данных областях применения - собранные датчиками данные используются для отображения текущего состояния окружающей среды. Так, например, при интеллектуальном отоплении, датчики измеряют климат в помещении. На основе этой информации и заданной пользователем целевого состояния микроклимата помещения, приложение может регулировать отопление, вентиляцию и кондиционирование воздуха. Сбор климатических данных, безусловно, имеет большое значение для обеспечения функционирования “умного” помещения, но в то же время создает риск нарушения конфиденциальности. Предотвратить утечку климатических данных из помещения может повышение кибербезопасности системы и ее надежности.

Если предположить, что злоумышленник способен обучить классификатор, который распознает определенные паттерны (действия людей, их присутствие), то возможными нарушениями конфиденциальности могут стать отслеживание присутствия и методов работы сотрудников в офисах, или раскрытие образа жизни в частных помещениях. Все эти ситуации представляют собой вторжение в внутреннюю деятельность предприятий и частную жизнь людей.

Данный проект направлен на оценку рисков конфиденциальности, возникающих при сборе данных микроклимата в помещении, и обосновании необходимости обеспечения качественной информационной безопасности системы. На защиту выносится гипотеза о том, что на основе данных о микроклимате можно определить наличие людей в помещении и вид их деятельности.

Актуальность исследования заключается в том, что в отличие от камер наблюдения и датчиков движения, человек в помещении с датчиками, отслеживающими микроклимат, не ожидает, что за ним будут наблюдать. Именно поэтому так важно доказать наличие этой угрозы на практике.

С целью обращения внимания на необходимость повышения надежности систем “умных” помещений, в рамках проекта создается нейросеть, способная предсказать наличие и количество людей в помещении и различать виды их деятельности. Это позволит не только получить ответ на вопрос, насколько реальны угрозы нарушения конфиденциальности при утечке климатических данных, но и повысить функциональность системы “умного” офиса, путем добавления возможности выявления паттернов посредством классификации.

Для того, чтобы реализовать нейронную сеть, необходимо выполнить следующие задачи:

1. Анализ способов построения и типов современных нейронных сетей.
2. Составление технического задания на проект.
3. Подготовка обучающего и тестового набора данных.
4. Построение и обучение нейронной сети.
5. Тестирование нейронной сети.
6. Оформление программной и проектной документации.

Работа выполнена в рамках приоритетного направления развития науки Пермского филиала НИУ ВШЭ “Исследование методов управления в киберфизических системах” [5].

Глава 1. Выбор способа построения и типа нейронной сети

В данной главе определяется связь между проблемой, которую необходимо решить, и задачей анализа данных - то есть способом решения поставленной проблемы с помощью возможностей машинного обучения. Выбираются средства построения нейронной сети (язык программирования, библиотека), а также архитектура нейронной сети.

## **1.1. Постановка задачи анализа данных**

В терминологии анализа данных, определение наличия людей в помещении и вида их деятельности на основе данных о микроклимате - это выявление паттернов в данных, что относится к задаче классификации. Классификация - одна из самых популярных задач в машинном обучении и заключается в отнесении объекта к одной из категорий на основании его признаков. Категории - т.е. классы, определены заранее (в машинном обучении задача классификации относится к разделу обучения в учителем). Также должно быть задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется обучающей выборкой. Классовая принадлежность остальных объектов неизвестна. На основе обучающей выборки строится алгоритм, способный классифицировать произвольный объект из исходного множества. Классифицировать объект — значит, указать номер (или наименование класса), к которому относится данный объект.

Так, относительно офисного помещения будут заданы три класса выходной переменной “наличие людей в помещении” и пять классов для выходной переменной “вид деятельности” (см. таблицы 1.1 - 1.2).

***Таблица 1.1. Классы выходной переменной “Наличие людей в помещении”***

|  |  |
| --- | --- |
| **№ класса** | **Расшифровка** |
| 0 | В помещении нет людей |
| 1 | В помещении находится 1 человек |
| 2 | В помещении находятся 2 человека |

***Таблица 1.2. Классы выходной переменной “Вид деятельности”***

| **№ класса** | **Расшифровка** |
| --- | --- |
| 0 | Нет информации (в комнате нет людей) |
| 1 | Человек в помещении сидит за столом и читает книгу |
| 2 | Человек стоит посередине комнаты, стараясь избегать движений |
| 3 | Человек медленно и беспорядочно перемещается по комнате |
| 4 | Человек сидит за столом и работает за компьютером |

Согласно эксперименту, в котором были собраны необходимые данные для исследования, предполагается, что, если в комнате находится больше, чем 1 человек, то они выполняют одни и те же действия (например, оба сотрудника работают за компьютером).

Заявка на проект (Приложение А) определяет, что необходимо реализовать нейронную сеть, способную обрабатывать сигналы, поступающие с датчиков, поэтому классификация будет реализована именно с помощью нейронной сети, обрабатывающей данные температуры, влажности, освещения. На основе этих входных данных нейросеть должна определять на выходе две переменные - “наличие людей в помещении” и “вид деятельности”.

Данные, использованные для решения задачи, описаны во второй главе отчета.

## **1.2. Обоснование выбора языка Python для построения нейронной сети**

Для начала необходимо обосновать выбор языка программирования, на котором была реализована нейронная сеть - Python. Рассмотрим преимущества и недостатки Python, R и MATLAB - одних из самых распространенных языков программирования для анализа данных.

Язык R создавался специально для анализа данных: запись конструкций языка понятна многим специалистам в области. Более того, многие функции, необходимые для анализа данных, являются встроенными, поэтому проверка статистических гипотез зачастую занимает лишь несколько строк кода. К преимуществам можно также отнести удобный репозиторий пакетов и обилие готовых тестов практически под все методы Data Science и машинного обучения. Язык предоставляет возможности эффективной работа с векторами и матрицами. Доступны пакеты для визуализации данных для различных задач (ggplot2, lattice, ggvis, googleVis, rCharts и др.).

К недостаткам языка R относят низкую производительность, что компенсируется наличием в системе пакетов, позволяющих повысить скорость работы (pqR, renjin, FastR, Riposte и др.). При работе с большими массивами данных рекомендуется использовать библиотеки data.table and dplyr. Язык R отличается от стандартных языков программирования своей узкоспециализированностью (например, индексация векторов начинается с единицы вместо нуля). R отлично подходит для решения статистических задач, но уступает в преимуществах в тех областях, где традиционно применяются языки общего назначения.

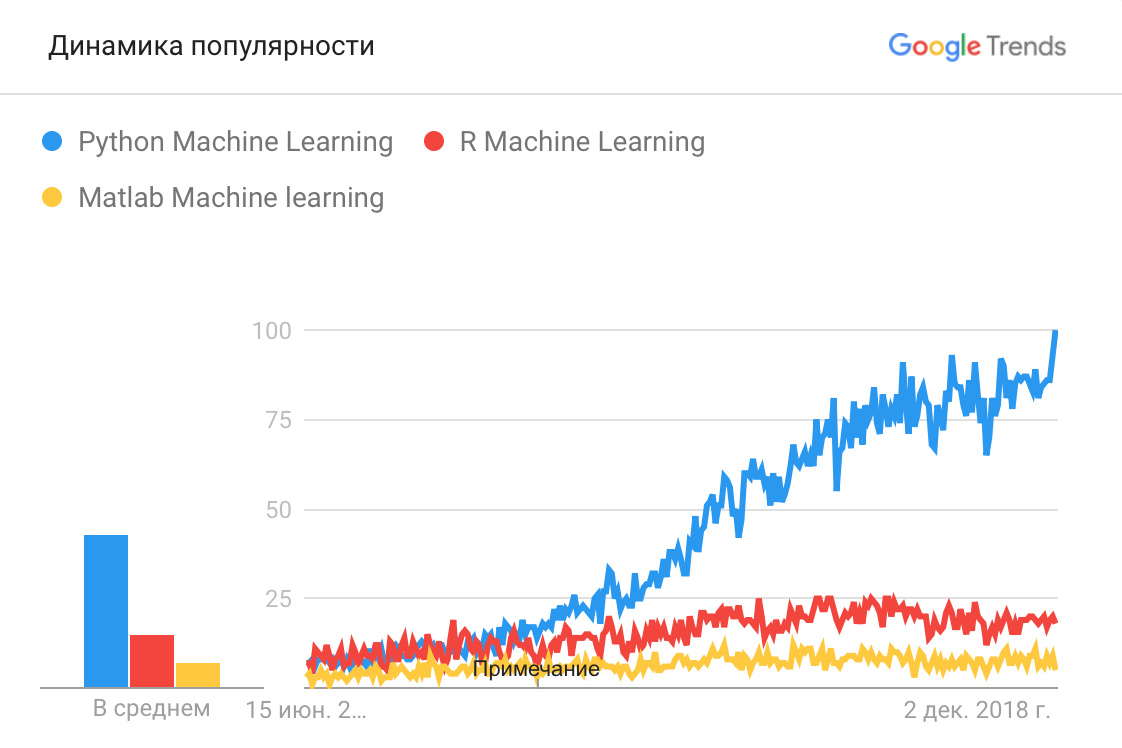
Широко используемый в науке и промышленности язык MATLAB – признанный язык для вычислений. Разработан и лицензирован MathWorks, компанией, созданной в 1984 году, которая занимается разработкой программного обеспечения. MATLAB идеально подходит для приложений, требующих сложных математических функций, также имеет ряд встроенных функций для визуализации данных. Используется во многих университетских курсах по физике, инженерии и прикладной математике. Как следствие широко применяется в этих областях.

К сожалению, инструмент MATLAB, для которого и был разработан язык, распространяется на коммерческой основе.

Язык Python выгодно отличается от перечисленных выше. Это универсальный многоцелевой язык: с его помощью можно осуществить не только обработку данных, но также их поиск и использование результата обработки в веб-приложении.

К недостаткам Python можно отнести нехватку альтернатив для многих библиотек R. Однако ситуация значительно улучшилась за последние годы: набор инструментов аналитиков сместился за последние два года в сторону библиотек на Python. Кроме того, начало программирования на Python облегчают такие сборки, как Anaconda. Сложностью в программировании на Python является то, что это язык с динамической типизацией. Несмотря на то, что это существенно ускоряет разработку программ, усложняется поиск некоторых трудно отслеживаемых ошибок, связанных с неправильным присваиванием различных данных одним и тем же переменным.

На рисунке 1.1 показано сравнение популярности поисковых запросов, связанных с Python, R и Matlab за последние пять лет [6]. Очевидно, что Python лидирует по популярности.



***Рисунок 1.1. Динамика популярности поисковых запросов, связанных с Python, MATLAB и R***

Именно язык Python чаще всего используют для построения нейронных сетей. Это связано с тем, что это язык программирования с низким порогом входа, написание программ облегчается использованием многочисленных библиотек, подходящих для создания нейросетей. В результате вокруг языка Python сложилось профессиональное сообщество по нейросетям.

Язык Python был освоен в рамках дисциплин “Технологии анализа данных” и “Методы и средства разработки информационных систем” в процессе обучения в магистратуре НИУ ВШЭ — Пермь, поэтому его использование в проекте позволит закрепить полученные навыки. Опыт работы в Python в совокупности с высокой степенью его применимости в построении нейронных сетей позволили сделать выбор в пользу использования этого языка программирования.

## 1.3. Обоснование выбора библиотеки Keras для построения нейронной сети

Существуют различные фреймворки для машинного обучения. Далее будут описаны некоторые из них, рассмотренные с целью определения, какой фреймворк наилучшим образом подойдет для решения поставленной задачи.

TensorFlow - один из лучших фреймворков глубокого обучения, в настоящее время используемый такими компаниями, как Airbus, Twitter, IBM и другими. Наиболее известный случай применения TensorFlow в сочетании с такими технологиями, как обработка естественного языка, распознавание речи/изображений/письма, прогнозирование - это Google Translate. TensorFlow поддерживает Python, C++ и R для создания моделей глубокого обучения. TensorFlow сопровождается полной документацией и инструкциями.

В связи с тем, что интерфейс TensorFlow сложен и это библиотека низкого уровня, на Python была написана библиотека Keras для обеспечения упрощенного интерфейса с целью быстрого создания прототипов путем построения эффективных нейронных сетей. Это открытая нейросетевая библиотека, представляет собой надстройку над фреймворками Deeplearning4j, TensorFlow и Theano. Нацелена на оперативную работу с сетями глубокого обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Библиотека Keras хорошо известна своей минималистичностью, она поддерживает как сверточные, так и рекуррентные сети, способные работать как на TensorFlow, так и на Theano. К недостаткам библиотеки относят нарушение универсальности кода: Keras 2.0 несовместима с первой версией, некоторые функции изменили название, некоторые были перемещены.

В целом, Keras используется, когда нужно быстро составить и протестировать сеть для решения конкретной задачи. В ситуации, когда нужны сложные модели, например, с нестандартными слоями, лучше использовать нижележащий фреймворк.

Еще один фреймворк для создания нейросетей - Torch. Если Keras написан на Python и хорошо интегрируется с массивной экосистемой данных Python, то Torch основан на Lua. Профессиональное сообщество, сформировавшееся вокруг Keras гораздо больше, чем у Torch. База пользователей Keras быстро растет, в то время как у Torch - сокращается, так как многие исследователи и организации переходят на TensorFlow/Keras. Документация Keras намного обширнее, в открытом доступе больше примеров кода и больше учебных ресурсов, помогающих начать работу.

На базе Torch была создана библиотека PyTorch — библиотека машинного обучения для языка Python с открытым исходным кодом. Эта библиотека разрабатывается преимущественно группой искусственного интеллекта Facebook. Она обеспечивает тензорные вычисления с GPU-ускорением. PyTorch предлагает API для решения прикладных задач, связанных с нейронными сетями.

PyTorch отличается от других фреймворков машинного обучения тем, что здесь не используются статические расчетные графы – определяемые заранее, сразу и окончательно – как в TensorFlow. Напротив, расчетные графы в PyTorch — динамические: при каждом вызове слоев в модели PyTorch динамически определяется новый расчетный граф. Этот граф создается имплицитно — библиотека сама записывает поток данных, идущих через программу, и связывает вызовы функций (узлы) вместе (посредством ребер) в расчетный граф. Данная библиотека используется для обработки естественного языка, поэтому ее применение в проекте необоснованно.

Еще одна из самых распространенных библиотек машинного обучения - Theano. Она обеспечивает тесную интеграцию с NumPy, использование CPU и GPU для повышения производительности, наличие встроенных механизмов оптимизации кода. Theano используется, когда необходимо произвести вычисления с большой точностью максимально быстро. Однако, в 2017 было отмечено [3], что разработка библиотеки Theano будет прекращена. Это позволяет сделать выбор в пользу постоянно обновляющегося фреймворка TensorFlow.

**Таблица 1.3. Результат сравнения библиотек для разработки нейронной сети**

| **Характеристика** | **TensorFlow/Keras** | **Theano/Keras** | **Torch/Pytorch** |
| --- | --- | --- | --- |
| Библиотека, специально разработанная для машинного обучения | 3 | 3 | 3 |
| Сохраняет актуальность на сегодняшний день | 3 | 1 | 3 |
| Основан на Python | 3 | 3 | 3 |
| Простота в использовании | 2 | 1 | 2 |
| Подходит для реализации поставленной задачи | 3 | 3 | 2 |
| Полнота документации | 3 | 2 | 1 |
| **Итого:** | **17** | 13 | 14 |

Таким образом, были рассмотрены такие фреймворки для машинного обучения, как TensorFlow и Theano c надстройкой Keras, Torch/PyThorch. На основании сравнительного анализа была составлена таблица, отражающая критерии выбора библиотеки для выполнения проекта. В результате такого анализа выбор был сделан в пользу библиотеки Keras, основанной на TensorFlow.

## 1.4. Обоснование выбора архитектуры нейронной сети

В данном разделе описаны наиболее популярные архитектуры нейронных сетей: перцептрон, свёрточные сети (CNN), рекуррентные сети (RNN), автоэнкодеры.

Сети прямого распространения (Feed forward neural networks, FF or FFNN) и перцептроны (perceptrons, P) очень просты — они передают информацию от входа к выходу (рис. 1.2.). Считается, что у нейронных сетей есть слои, каждый из которых состоит из входных, скрытых или выходных нейронов. Нейроны одного слоя между собой не связаны, при этом каждый нейрон этого слоя связан с каждым нейроном соседнего слоя. FFNN обычно обучают методом обратного распространения ошибки, подавая модели на вход пары входных и ожидаемых выходных данных. Под ошибкой обычно понимаются различные степени отклонения выходных данных от исходных (например, среднеквадратичное отклонение или сумма модулей разностей). При условии, что сеть обладает достаточным количеством скрытых нейронов, теоретически она всегда сможет установить связь между входными и выходными данными. На практике использование сетей прямого распространения ограничено, и чаще они используются совместно с другими сетями.

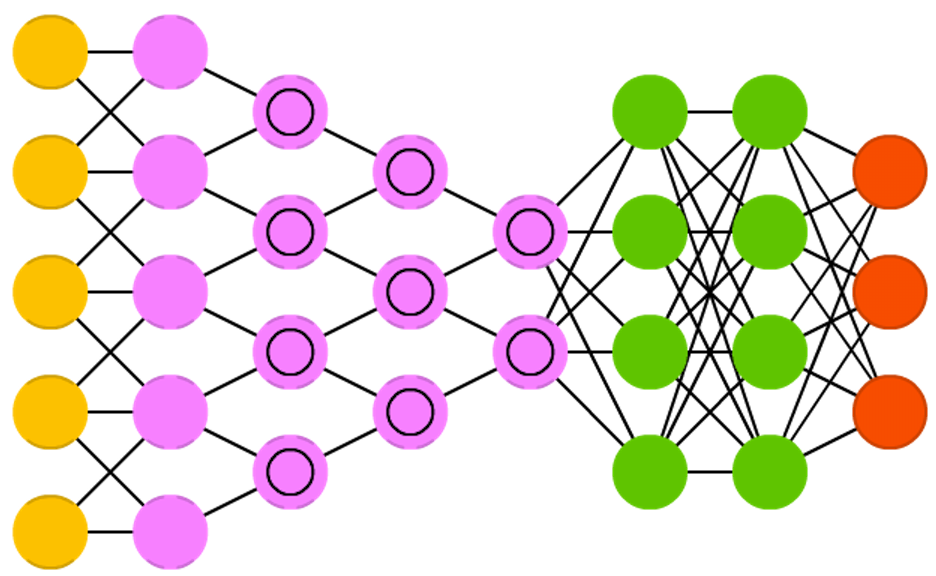


**Рисунок 1.2. Сети прямого распространения**

Свёрточные сети [4] используются для поиска объектов на фото и видео, распознавания лиц, переноса стиля, генерации и дорисовки изображений, создания эффектов и улучшения качества фотографий. Сегодня CNN применяют там, где есть изображения или видео. идея свёрточных нейронных сетей заключается в чередовании свёрточных слоёв (convolution layers) и субдискретизирующих слоёв (subsampling layers или pooling layers, слоёв подвыборки). Структура сети — однонаправленная (без обратных связей), принципиально многослойная (рис. 1.3). Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов — любая, по выбору исследователя.

На практике к концу CNN прикрепляют FFNN для дальнейшей обработки данных. Такие сети называются глубокими (DCNN), но названия их обычно взаимозаменяемы.

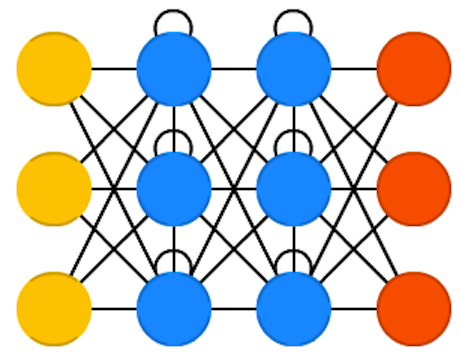
Так как этот тип сетей используется для распознавания и классификации изображений, он не будет рассмотрен более подробно в рамках данного проекта.



***Рисунок 1.3. Глубокая свёрточная нейронная сеть***

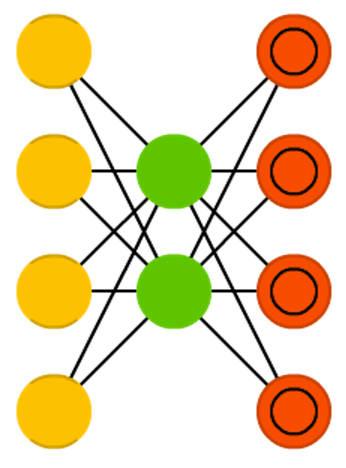
Рекуррентные нейронные сети — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Это те же сети прямого распространения, но со смещением во времени: нейроны получают информацию не только от предыдущего слоя, но и от самих себя в результате предыдущего прохода. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где целостное разбито на части, например: распознавание рукописного текста или распознавание речи, машинный перевод текстов, компьютерный синтез речи. Было предложено много различных архитектурных решений для рекуррентных сетей. В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью LSTM) и управляемый рекуррентный блок (GRU).

У RNN есть один большой недостаток — это проблема исчезающего (или взрывного) градиента: в зависимости от используемой функции активации информация со временем теряется, так же как и в очень глубоких сетях прямого распространения.



***Рисунок 1.4. Рекуррентная нейронная сеть***

Автоэнкодеры (Autoencoders, AE) — похожи на FFNN, это скорее другой способ использования FFNN, а не принципиально новая архитектура. Основная идея автоэнкодеров — автоматическое кодирование (как при сжатии, а не при шифровании) информации. Сеть напоминает по форме песочные часы, так как скрытый слой меньше, чем входной и выходной; к тому же она симметрична относительно средних слоев (одного или двух, в зависимости от четности/нечетности общего количества слоев). Самый маленьких слой почти всегда средний, в нем информация максимально сжата. Все, что расположено до середины — кодирующая часть, выше середины — декодирующая, а в середине — код. AE обучают методом обратного распространения ошибки, подавая входные данные и задавая ошибку равной разницу между входом и выходом.

****

***Рисунок 1.5. Автоэнкодер***

Простейшая архитектура автоэнкодера — сеть прямого распространения, без обратных связей, наиболее схожая с перцептроном и содержащая входной слой, промежуточный слой и выходной слой. В отличие от перцептрона, выходной слой автокодировщика должен содержать столько же нейронов, сколько и входной слой.

Основными практическими приложениями автокодировщиков остаются уменьшение шума в данных, а также уменьшение размерности многомерных данных для визуализации. С определёнными оговорками, касающимися размерности и разрежённости данных, автокодировщики могут позволять получать проекции многомерных данных, которые оказываются лучше тех, что даёт метод главных компонент либо какой-либо другой классический метод.

На основании анализа областей использования различных архитектур нейросетей, был сделан вывод, что для классификации числовых показателей с датчиков подойдет самая простая архитектура - перцептрон.

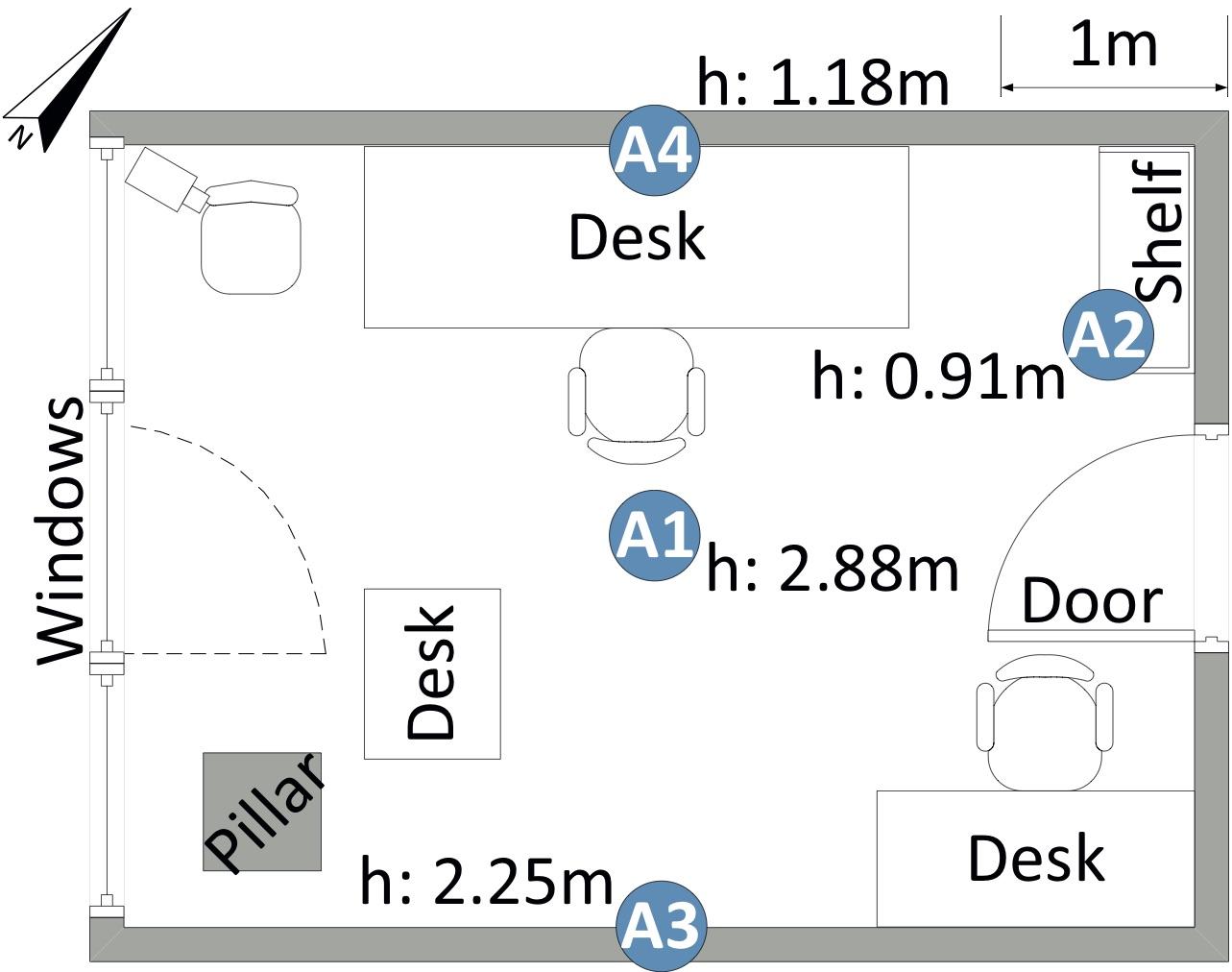
# Глава 2. Предварительная обработка данных для нейронной сети

Данные, на которых обучается нейросеть, требуют тщательного отбора и подготовки. Так, в данной главе описывается источник данных, их характеристики и подготовка к обучению и тестированию нейросети.

## 2.1. Описание данных с датчиков, установленных в офисном помещении

Для выполнения проекта необходимы данные с датчиков, установленных в офисном помещении. Обзор литературы позволил найти исследование, проведенное немецкими учеными и направленное на изучение вопросов безопасности систем “умного” отопления в помещениях [1]. С целью дать возможность другим исследователям продолжить работу по анализу климатических данных в помещении, авторы исследования предоставили данные по замерам температуры, влажности, степени освещенности, состояния дверей и окон в офисном помещении в открытом доступе на GitHub [2].

Сбор данных осуществлялся на протяжении двух месяцев с марта по апрель 2016 года. Так, в офисном помещении площадью 16.5 квадратных метров были установлены 4 датчика на разных высотах. Расположение датчиков и план помещения показаны на рисунке 2.1.



***Рисунок 2.1. План помещения и расположение датчиков. Высоты, на которых установлены датчики: А1 на высоте 2.88 м, А2 - 0.91 м, А3 - 2.25 м, А4 - 1.18 м***

Следует отметить, что результат обучения нейронной сети по таким данным будет применим только к помещениям, схожим по техническим характеристикам с экспериментальным. Для применения нейросети к другим помещениям, например, большим по площади или с другим расположением датчиков, потребуется переобучить созданную нейросеть. Тем не менее, данное помещение (рис. 2.1) по своим размерам и планировке можно отнести к типичному офисному помещению, что делает результаты исследования широко применимыми, например, в создании систем “Умного офиса”.

Данные представлены в 60 файлах формата .csv и содержит серию последовательных измерений климата в помещении. Необходимо отметить, что каждый файл содержит измерения сразу по нескольким датчикам. Все данные в файле - в формате float64.

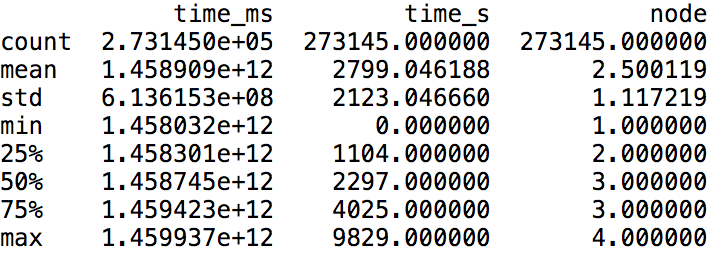
Каждое наблюдение в файле состоит из следующих измерений:

* номер наблюдения в предалах файла;
* абсолютная метка времени в миллисекундах;
* относительная метка времени в секундах;
* ID узла (датчика);
* температура в °C;
* относительная влажность в %;
* показатель датчика освещенности 1 - длина волны в нм;
* показатель датчика освещенности 2 - длина волны в нм;
* количество человек в помещении (0, 1, 2);
* активность в помещении (0 = нет данных (никого нет в помещении), 1 = человек читает за столом, 2 = человек стоит в центре комнаты, 3 = человек перемещается по комнате, 4 = человек работает за компьютером);
* состояние двери (0 = закрыто, 1 = открыто);
* состояние окна (0 = закрыто, 1 = открыто).

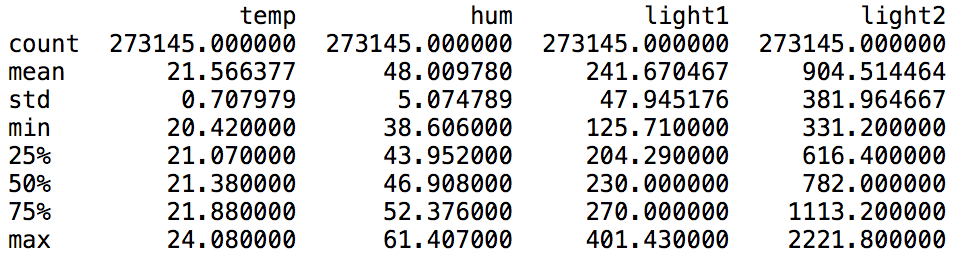
Для удобства работы с данными, колонкам были присвоены следующие наименования: "id", "time\_ms", "time\_s", "node", "temp", "hum", "light1", "light2", "occup", "act", "door", "win".

## 2.2. Подготовка данных для обучения нейронной сети

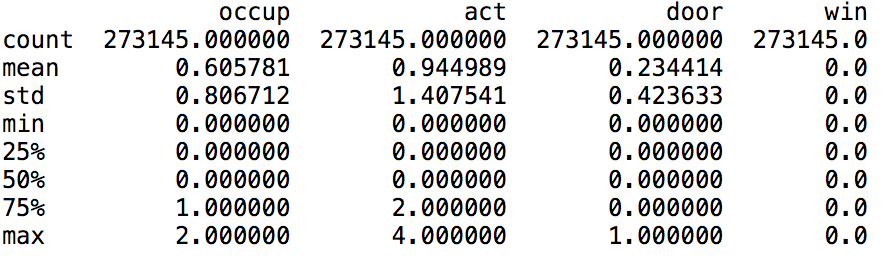
Прежде всего, данные были импортированы из всех 60 файлов в один объект типа ndarray, который был преобразован в dataframe для удобства работы с наблюдениями. Статистическое описание всех данных показано на рисунках 2.2 - 2.4.



***Рисунок 2.2. Статистические характеристики по колонкам “time\_ms”, "time\_s", "node"***



***Рисунок 2.3. Статистические характеристики по колонкам "temp", "hum", "light1", "light2"***



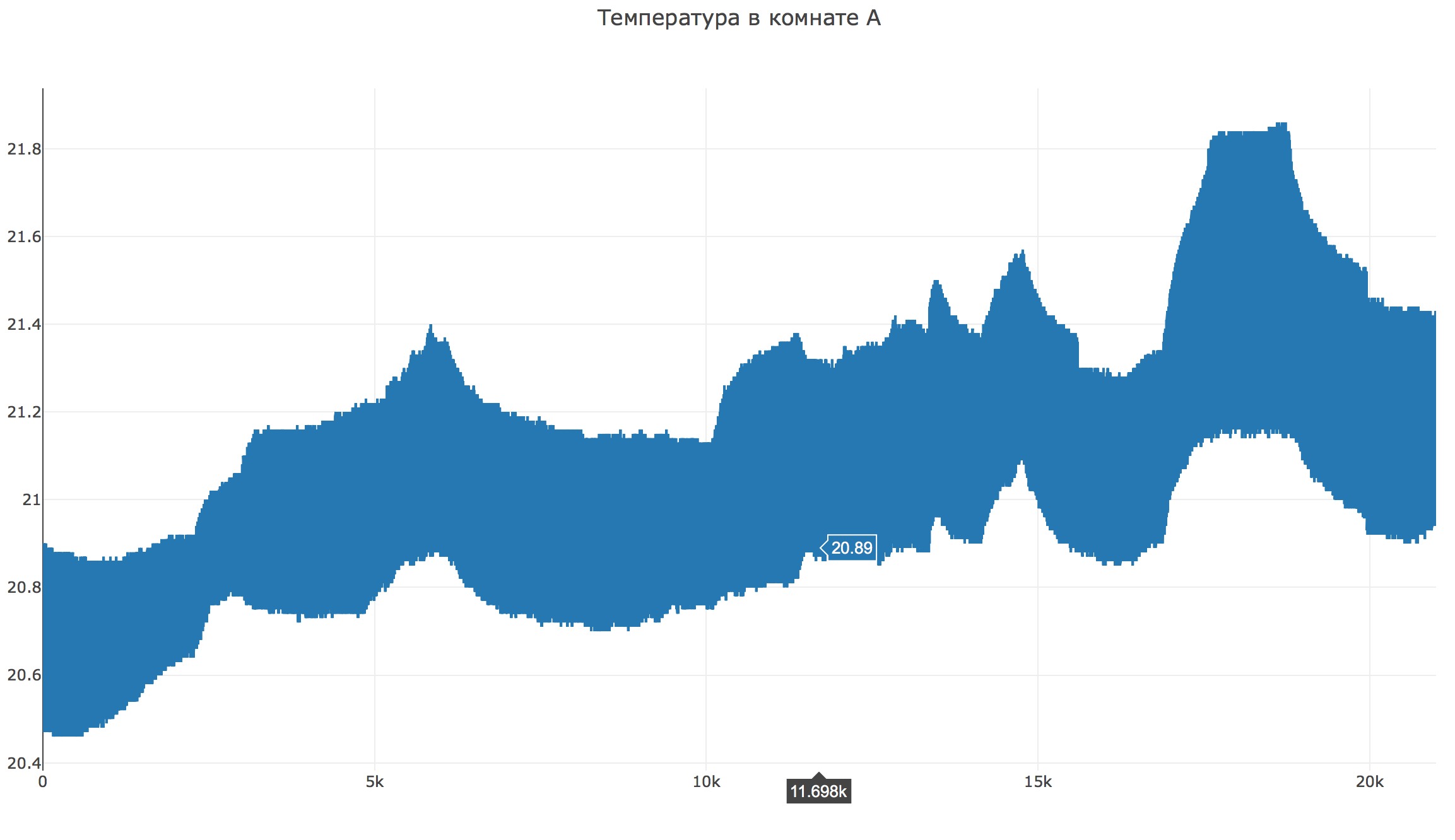
***Рисунок 2.4. Статистические характеристики по колонкам "occup", "act", "door", "win"***

Затем данные были разделены на два объекта типа dataframe - один для зависимых переменных (количество людей в помещении и вид деятельности), второй для независимых переменных:

Y = dataframe.copy()[["occup","act"]]

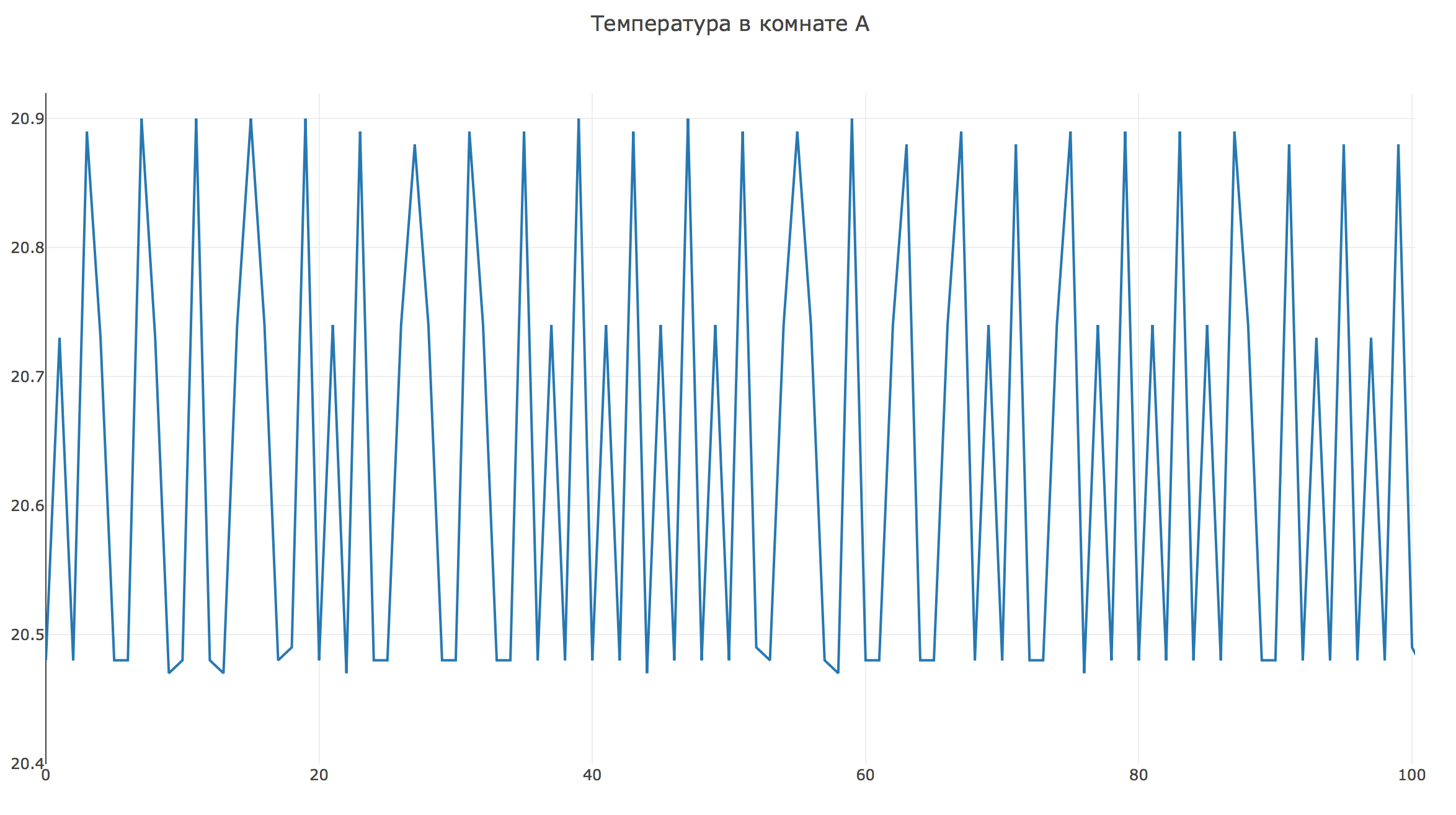
X = dataframe.copy().drop(columns = ["occup","act"])

Визуальный анализ данных позволил понять, за какие промежутки времени датчики фиксируют измерения. Прежде всего был визуализирован весь ряд температуры (рис. 2.5). График выглядит сплошным, поскольку визуализировано большое количество измерений: 21 000 строк данных из 273 145.



***Рисунок 2.5. Частичная визуализация ряда температуры в помещении***

Если ограничиться первыми 100 измерениями, получится график, показанный на рисунке 2.6. Он соответствует приблизительно одной минуте наблюдений (измерения снимаются чаще, чем раз в секунду). Видим, что фиксируются изменения температуры с точностью до десятых градуса.



***Рисунок 2.6. Визуализация измерений температуры, полученных на протяжении минуты***

Для обучения нейросети данные следует подавать на вход в нормализованном виде, что было достигнуто с помощью MinMaxScaler:

scaler = MinMaxScaler(feature\_range=(-0.5, 0.5))

X\_scaled = pd.DataFrame(scaler.fit\_transform(X), columns = X.columns)

Нейронная сеть будет обучена плохо, если распределения значений признаков в обучающей и тестовой выборке имеют разные характеристики. Для того, чтобы избежать этой ситуации, используется параметр stratify метода train\_test\_split из библиотеки scikit-learn.

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X\_scaled, Y["occup"], test\_size=0.33, random\_state=42, stratify = Y)

Проверить равномерность разбиения выборки и долю присутствия каждого класса в тестовой и обучающей выборках можно следующим образом:

unique\_elements, counts\_elements = np.unique(np.array(Y\_train["occup"]), return\_counts=True)

print("Train:", unique\_elements, counts\_elements)

unique\_elements, counts\_elements = np.unique(np.array(Y\_test["occup"]), return\_counts=True)

print("Test:", unique\_elements, counts\_elements)

Код возвращает следующий результат: Train: [0. 1. 2.] [109842 35468 37697], Test: [0. 1. 2.] [54101 17470 18567]. Видим, что распределение равномерное. Тоже самое можно повторить и для второго целевого класса “вид деятельности”.

Следует отметить, что данные были разделены на обучающую и тестовую выборки дважды. В первый раз - для предсказания количества человек в комнате, второй раз - для предсказания вида деятельности в комнате. В итоге было обучено две нейросети, определяющие соответственно первую или вторую зависимую переменную. Такое решение было принято на основе ранее выбранной архитектуры нейронной сети. Реализация нейронной сети, выполняющей сложную (не бинарную) классификацию - достаточно трудоемкий процесс. Нейронная сеть, реализующая мультиклассовую классификацию для двух выходных переменных еще более сложна в реализации. Реализация такой нейросети может быть запланирована в качестве дальнейших исследований.

# Глава 3. Построение и обучение нейронной сети для обработки сигналов

В данной главе описан процесс построения нейронной сети, настройка процесса обучения, результат оценки качества полученной нейронной сети.

## 3.1. Построение нейронной сети

Модель в библиотеке Keras строится либо с использованием класса Sequential, который представляет собой линейный набор слоев, либо с помощью более гибкого класса Model. В работе использовалась простая последовательная модель, поскольку выбранная архитектура сети представляет собой линейный стек слоев. Рассмотрим процесс построения нейронной сети, определяющей количество людей в помещении.

Построение модели начинается с ее инициирования:

from keras.models import Sequential

from keras.layers import Dense

model = Sequential([.....])

model.summary()

Конструктор принимает массив слоёв. Поскольку создается стандартная сеть прямой передачи данных, нам нужен только слой класса Dense, являющийся простым полностью связанным слоем. Было добавлено 3 слоя - входной, скрытый и выходной слои:

model = Sequential([Dense(64, activation='relu',

Dense(64, activation='relu'),

Dense(3, activation='softmax'),])

Первые два слоя имеют по 64 узла (нейрона) и используют функцию активации ReLU. Последний слой представляет собой выходной слой Softmax с 3 узлами, по одному для каждого класса “количество людей в помещении”. При построении нейронной сети для определения вида деятельности, в последнем выходном слое необходимо будет указать 5 нейронов, поскольку количество классов для зависимой переменной “вид деятельности” равен 5.

Последнее, что необходимо указать, это размерность входных данных сети. Это можно сделать, указав параметр input\_shape в первом слое:

model = Sequential([Dense(64, activation='relu', input\_shape=(9,)),

Dense(64, activation='relu'),

Dense(3, activation='softmax'),])

Указание размерности входных данных позволяет модели автоматически определять форму входных данных для последующих слоев. На этом определение модели завершается, далее необходимо настроить процесс обучения.

На этапе компиляции определяются 3 ключевых фактора:

1. Оптимизатор. По умолчанию этот параметр задает использование оптимизационного алгоритма Adam. В библиотеке есть и другие оптимизаторы, применение которых можно рассмотреть в дальнейшем в экспериментировании с качеством предсказания сети.
2. Функция потерь. Поскольку в качестве функции активации на выходном слое была указана Softmax, в качестве функции потерь будет использована функция Cross-Entropy loss, в библиотеке Keras доступны binary\_crossentropy (для 2 классов) и categorical\_crossentropy (для ситуаций, когда классов больше 2). Поскольку в целевой переменной 3 класса, была выбрана функция categorical\_crossentropy.
3. Метрики качества. Поскольку решается задача классификации, используется метрика accuracy.

Определение компиляции модели:

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'],)

## 3.2. Обучение нейронной сети

Обучение модели в Keras состоит только из вызова функции fit() и указания некоторых параметров. Существует множество возможных параметров, но в проекте вручную были указаны только основные:

1. Обучающая выборка (наблюдения и указания классов к ним).
2. Параметр epochs - количество итераций обучения по всему набору данных.
3. Параметр batch\_size - количество наблюдений, используемых в функции оптимизации во время обучения.

history = model.fit(

np.array(X\_train),

to\_categorical(Y\_train),

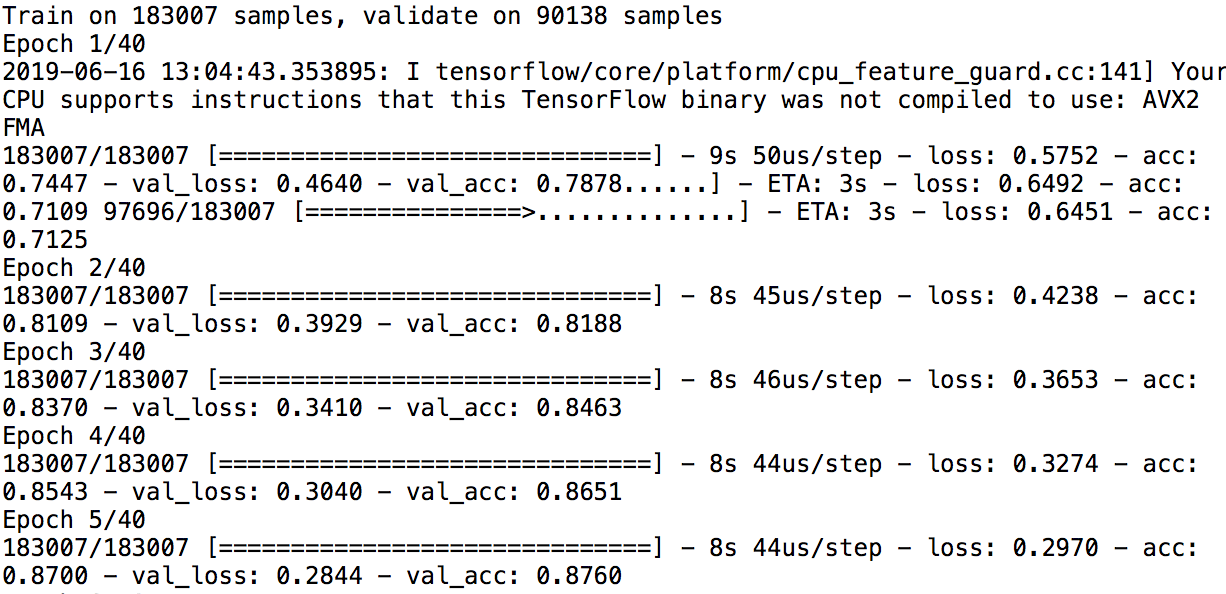
epochs=40,

batch\_size=32,

validation\_data=(np.array(X\_test), to\_categorical(Y\_test)))

Метод to\_categorical для меток классов используется, поскольку функция ошибки categorical\_crossentropy ожидает, что целевой переменной обучения будут 3-мерные векторы, поскольку в выходном слое с функцией активации Softmax указано 3 нейрона. Без преобразования значения целевой переменной - одно целое число, представляющее класс для каждого наблюдения. Метод to\_categorical() превращает массив целых чисел классов в массив векторов. Так, Y\_train, равный 0 станет равен [1 0 0], Y\_train, равный 1 станет равен [0 1 0], Y\_train, равный 2 станет равен [0 0 1].

На данном этапе модель готова. Процесс обучения нейронной сети выглядит так, как показано на рисунке 3.1:



***Рисунок 3.1. Процесс обучения нейронной сети***

Видим, что уже после пяти итераций обучения точность результата классификации нейронной сетью составила 87 % (рис. 3.1). По мере увеличения итераций точность будет повышаться.

## 3.3. Оценка качества работы нейросети

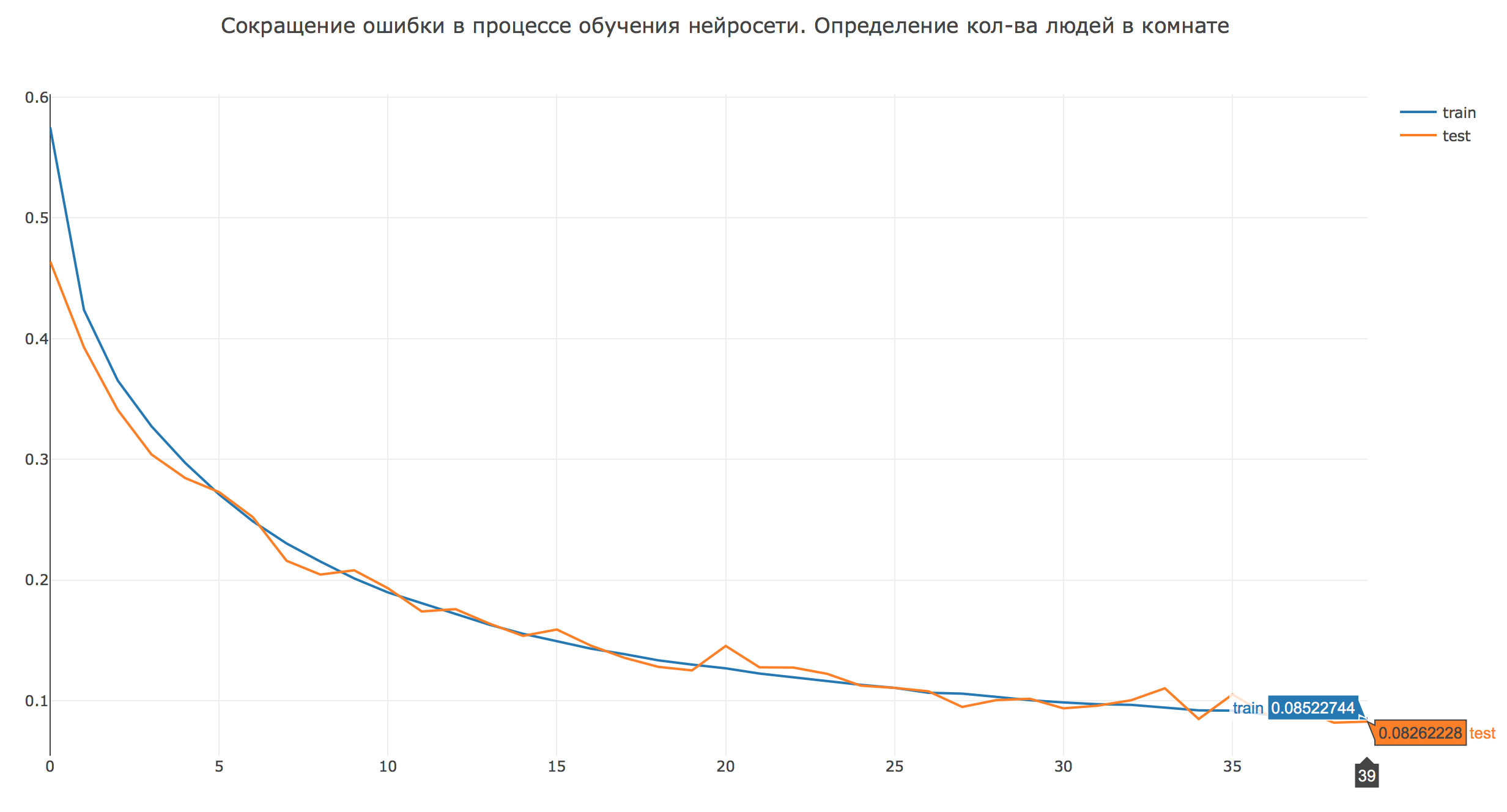
Для оценки качества предсказания используется метод evaluate(), возвращающий массив, содержащий ошибку и метрику, указанную при компиляции модели.

\_, score\_test = model.evaluate(np.array(X\_test), to\_categorical(Y\_test), verbose=1)

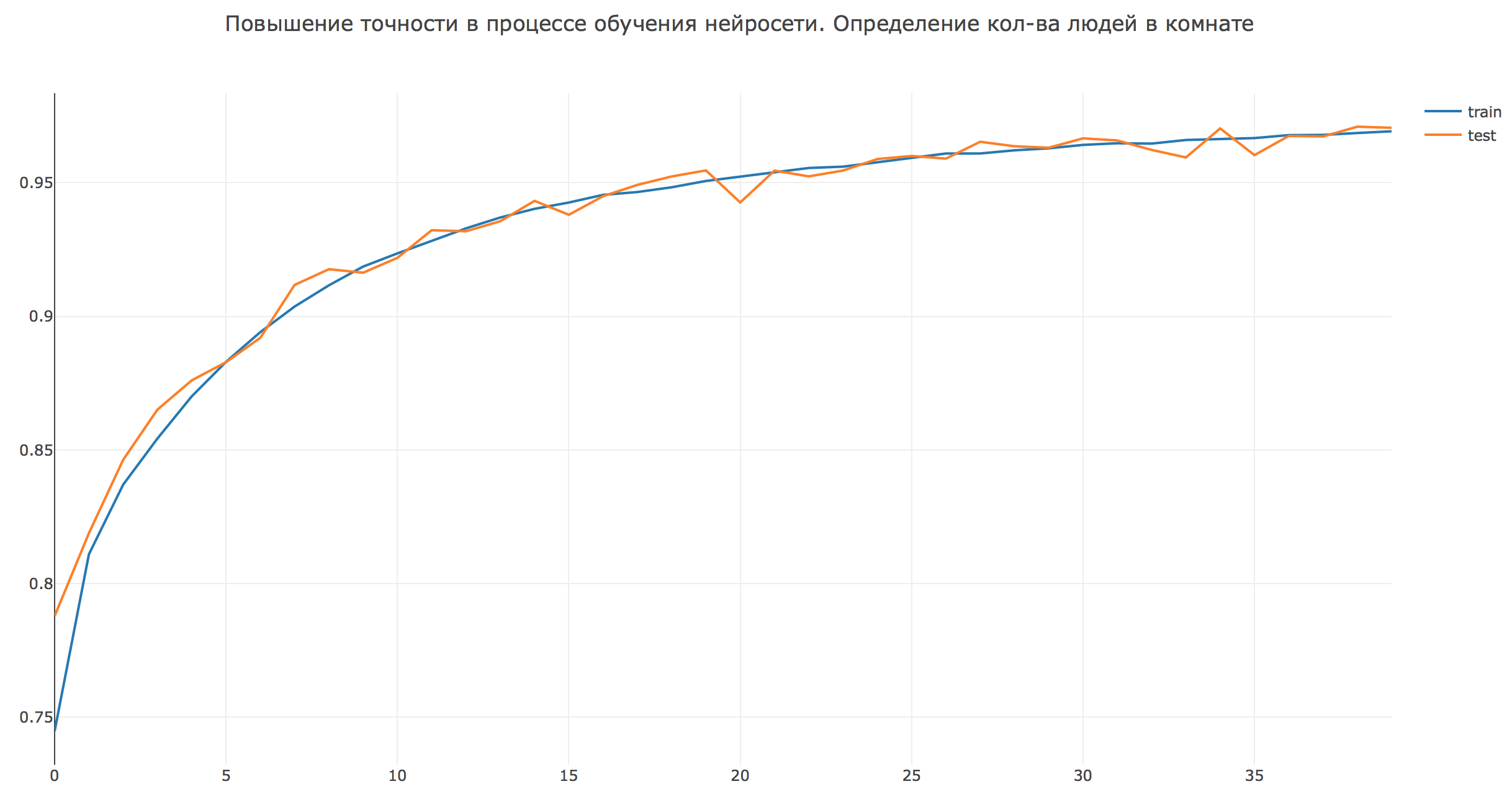
\_, score\_train = model.evaluate(np.array(X\_train), to\_categorical(Y\_train), verbose=1)

print('Train: %.3f, Test: %.3f' % (score\_train, score\_test))

Вызов этого метода показал, что точность определения количества человек в помещении на обучающей выборке составила 97,2%, а на тестовой 97,1%.

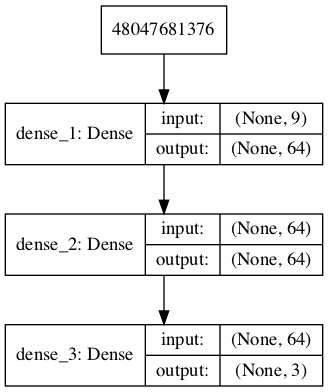


***Рисунок 3.2. Сокращение ошибки в процессе обучения нейронной сети определять количество людей в помещении***



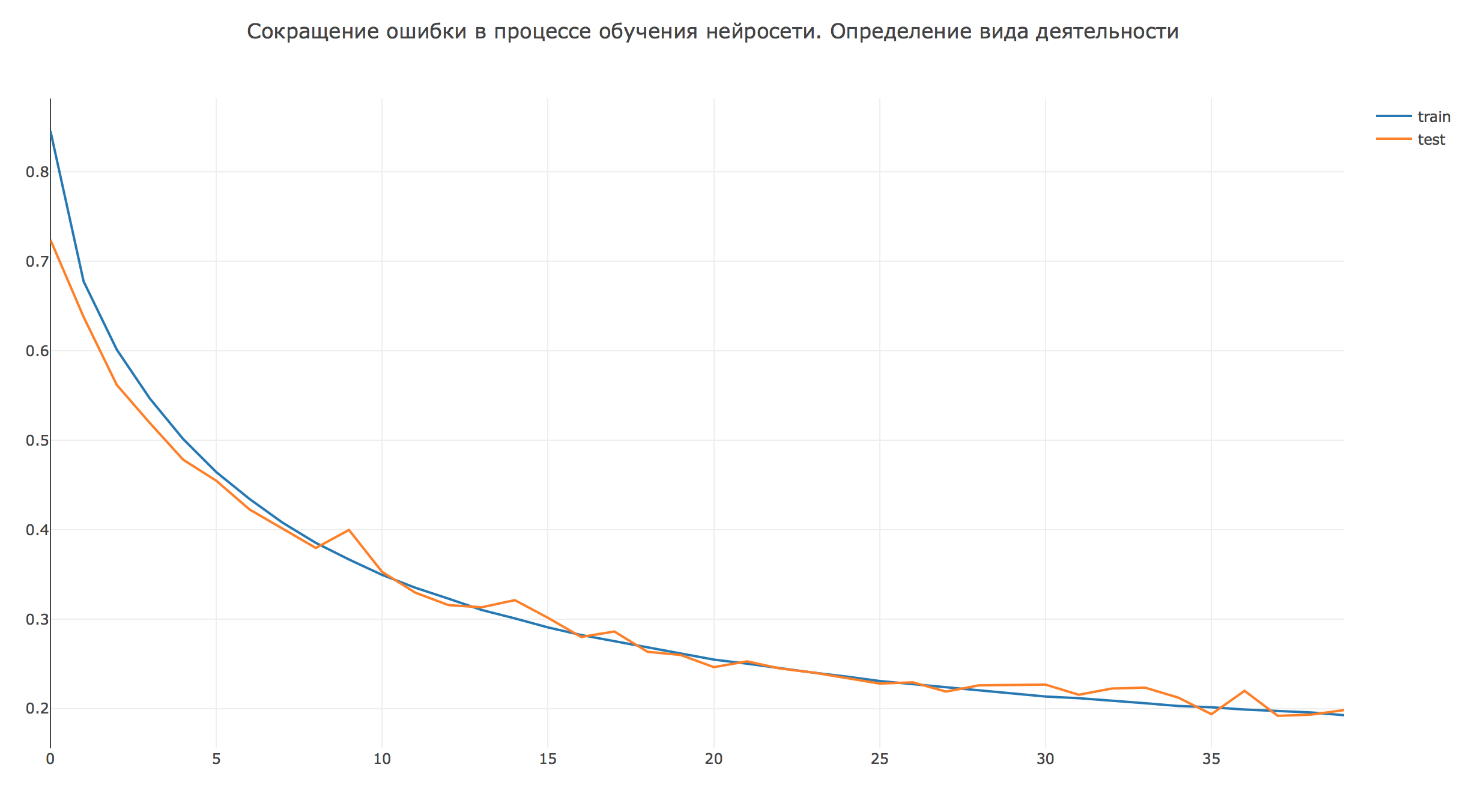
***Рисунок 3.3. Повышение точности классификации в процессе обучения нейронной сети определять количество людей в помещении***

Архитектура нейронной сети, определяющей количество человек в помещении показана на рисунке 3.4.

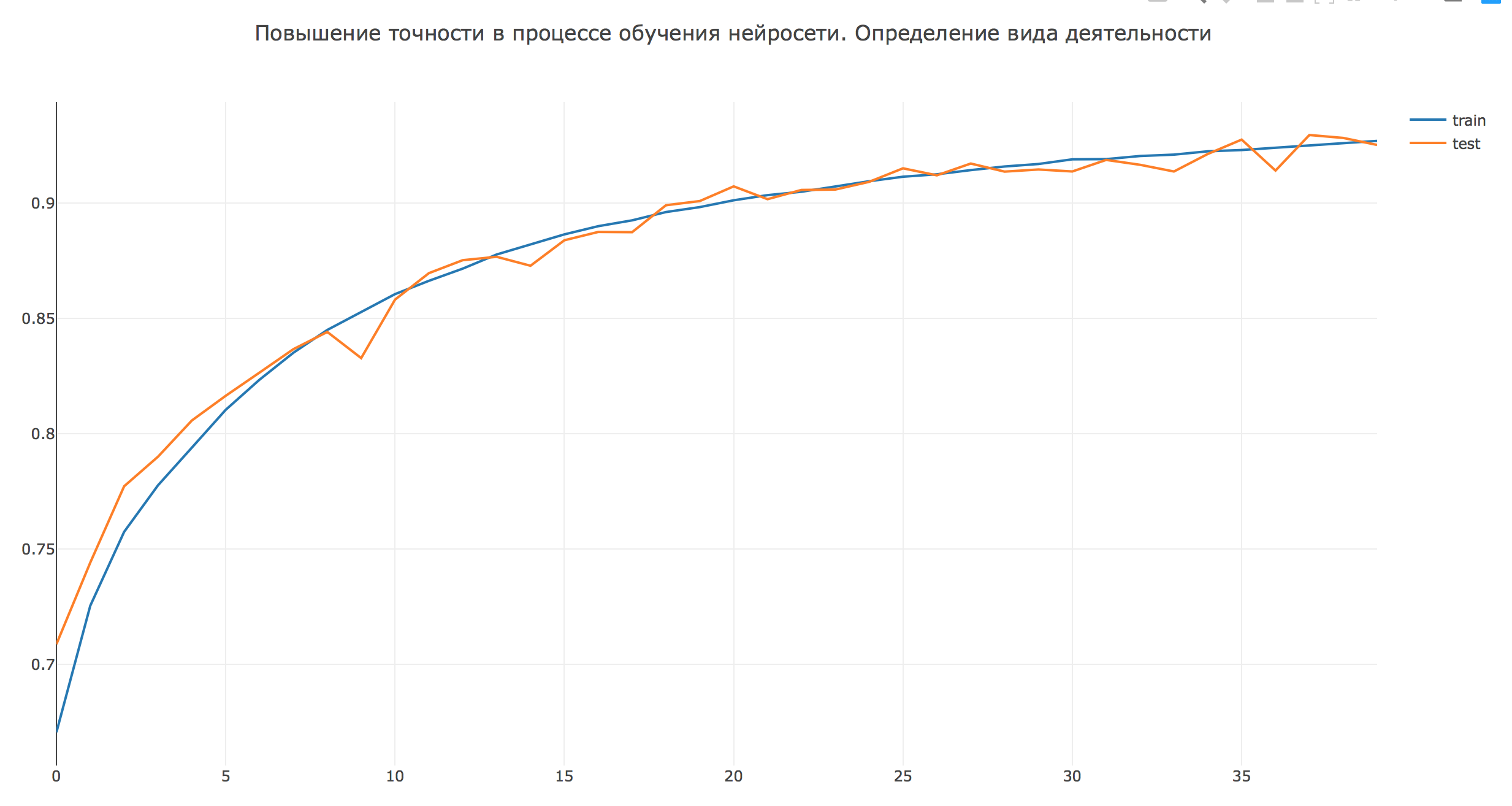


***Рисунок 3.4. Архитектура нейронной сети, определяющей количество человек в помещении***

Вызов метода evaluate() для нейросети, обученной определять вид деятельности в помещении показал, что точность определения этой целевой переменной на обучающей выборке составила 92,9%, а на тестовой 92,5%. Графики сокращения ошибки и повышения точности второй нейронной сети показаны на рисунках 3.4 -3.5.

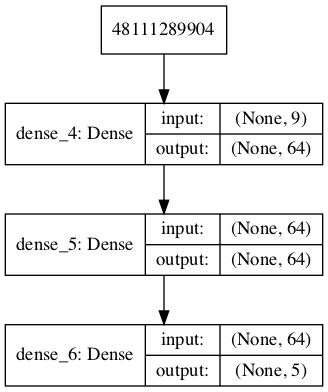


***Рисунок 3.5. Сокращение ошибки в процессе обучения нейронной сети определять вид деятельности***



***Рисунок 3.6. Повышение точности классификации в процессе обучения нейронной сети определять вид деятельности***

Архитектура нейронной сети, определяющей вид деятельности в помещении, показана на рисунке 3.7.



***Рисунок 3.7. Архитектура нейронной сети, определяющей вид осуществляемой деятельности в помещении***

# Заключение

Таким образом, были построены две нейросети, обрабатывающие показания с датчиков, установленных в офисном помещении. Обучение нейронных сетей и оценка их качества показала, что точность определения количества человек в помещении может составлять 97,1%, а точность определения вида деятельности - 92,5%.

Точность прогноза повышается, в зависимости от количества данных, используемых для обучения. Результаты исследования свидетельствуют о том, что даже утечка таких "незаметных" данных, как температура и относительная влажность, может серьезно нарушить неприкосновенность частной жизни и конфиденциальность на предприятиях. Значит, опасения по поводу нарушения конфиденциальности в результате утечки климатических данных в помещении вполне оправданы.

Результаты выполнения проекта могут быть использованы для расширения функционала системы “Умный офис”, поскольку результат работы нейронной сети позволяет определять паттерны в данных.

# Список литературы

1. Armknecht F., Benenson Z., Morgner P., Müller C., Riess C. *Privacy implications of room climate data.* Journal of Computer Security. 2018. pp. 1–24 P. Morgner, C. Müller, M. Ring, B. Eskofier, C. Riess, F. Armknecht, Z. Benenson: Privacy Implications of Room Climate Data. To appear in the Procedings of the European Symposium on Research in Computer Security (ESORICS) 2017, Oslo, Norway, September 11-13, 2017.
2. Armknecht F., Benenson Z., Morgner P., Müller C., Riess C. *Privacy implications of room climate data.* Данные, использованные для исследования. URL: <https://github.com/IoTsec/Room-Climate-Datasets> (дата обращения 15.06.19).
3. Comparison of AI Frameworks. URL: <https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch#torch> (дата обращения 15.06.19).

1. [Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation](http://deeplearning.net/tutorial/lenet.html). *DeepLearning 0.1*. LISA Lab. URL: <http://deeplearning.net/tutorial/lenet.html> (дата обращения 15.06.19).
2. Неганова Э.А., Марквирер В.Д., Кычкин А.В. Организация киберфизической системы интеллектуального офисного здания // Материалы ежегодной межвузовской научно-технической конференции студентов, аспирантов и молодых специалистов имени Е.В.Арменского (Москва). 2019. c. 226-227
3. Сравнение MATLAB, Python и R. URL: <https://trends.google.com/trends/explore?date=today%205-y&q=Python%20Machine%20Learning,R%20Machine%20Learning,Matlab%20Machine%20learning>

# Приложение

## Приложение А. Заявка на проект

СОГЛАСОВАНО

И.о. академического руководителя образовательной программы   
«Информационная аналитика в управлении предприятием»

А.И. Дерябин

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**ЗАЯВКА НА ПРОЕКТ**

**Название проекта: «Разработка нейронной сети для обработки сигналов датчиков»**

**Заказчик проекта:** кафедра информационных технологий в бизнесе, группа исследования киберфизических систем.

**Тип проекта:** практико-ориентированный (прикладной).

**Планируемые результаты проекта**

Образовательные: в результате выполнения проекта студенты формируют следующие компетенции - ОПК-1, ОПК-2, УК-1, УК-2, ОПК-5.

Проектные: готовая нейронная сеть для обработки сигналов с датчиков.

**Сроки и условия реализации проекта**

Плановые сроки начала: апрель 2019 года;

Плановые сроки окончания: июнь 2019 года.

**Содержание** **проекта:**

В рамках выполнения проекта должны быть выполнены следующие задачи:

* анализ способов построения и типов современных нейронных сетей;- составление технического задания на проект;
* выбор типа и способа построения нейронной сети;
* построение нейронной сети;
* подготовка обучающего и тестового набора данных;
* выполнение обучения нейронной сети;
* тестирование нейронной сети;
* оформление программной и проектной документации;
* представление результата проекта заказчику.

Отчетность по проекту должна включать:

* проектную заявку;
* техническое задание на проект;
* отчет о выполнении проекта;
* программу, реализующую нейронную сеть.

**Форма представления итогового результата:** программа нейронной сети.

Руководитель проекта доцент кафедры информационных технологий в бизнесе, к.т.н., доцент, Викентьева Ольга Леонидовна.

## Приложение Б. Программа нейронной сети

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Jun 15 00:21:35 2019

@author: elviraneganova

"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from numpy import genfromtxt

import plotly

import plotly.graph\_objs as go

from keras.utils import to\_categorical

filepath = "/Users/elviraneganova/Google Drive/ИАУП 18/Проект нейронная сеть/"

#%%

def plotly\_df(df, path, title, names) :

"""Визуализация интерактивного графика"""

data = []

for column, i in zip(df.columns, range(0, len(names))):

trace = go.Scatter(

x = df.index,

y = df[column],

mode = 'lines',

name = names[i]

)

data.append(trace)

layout = dict(title = title)

fig = dict(data = data, layout = layout)

plotly.offline.plot(fig, filename=path+title +'.html', auto\_open=False) #сохраняет интерактивный график в html файл в папке с кодом

#%%

#Импортируем данные

adress = filepath + "Данные/datasets-location\_A/room\_climate-location\_A-measurement"

data = []

for i in range (1, 61): #60 наборов данных

if len(str(i)) == 1 :

num = "0"+str(i)

else :

num = str(i)

num = num + ".csv"

data.append(genfromtxt((adress+num),delimiter=','))

data\_train\_test =np.vstack((data[0], data[1]))

for i in range(2, len(data)) :

data\_train\_test = np.vstack((data\_train\_test, data[i]))

print("Размерность всех данных:", data\_train\_test.shape)

#с датафреймом удобнее работать

dataframe = pd.DataFrame(data\_train\_test, columns = ["id", "time\_ms", "time\_s", "node", "temp", "hum", "light1", "light2", "occup", "act", "door", "win"]).drop(columns = ["id"])

dataframe.shape

#%%

#X - независимые переменные, Y - зависимые переменные

Y = dataframe.copy()[["occup","act"]]

X = dataframe.copy().drop(columns = ["occup","act"])

#%%

#Визуальный анализ данных

plotly\_df(df = X[['temp']].iloc[:21000], path = filepath + "Данные/Графики/", title = 'Температура в комнате А', names = ['temp'])

#%%

from sklearn.preprocessing import MinMaxScaler

#Нормализует данные, приводя их к одной шкале (feature\_range)

scaler = MinMaxScaler(feature\_range=(-0.5, 0.5))

X\_scaled = pd.DataFrame(scaler.fit\_transform(X), columns = X.columns)

X\_scaled.shape

#%%

#ОПРЕДЕЛЯЕМ КОЛИЧЕСТВО ЧЕЛОВЕК В КОМНАТЕ

#Делим ровно выборки

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X\_scaled, Y["occup"], test\_size=0.33, random\_state=42, stratify = Y)

#Параметр stratify сохраняет пропорции зависимой переменной в тестовой и

#обучающей выборках такими же, как и в исходной выборке

#%%

from keras.models import Sequential

from keras.layers import Dense

model = Sequential([

Dense(64, activation='relu', input\_shape=(9,)),

Dense(64, activation='relu'),

Dense(3, activation='softmax'),

])

model.summary()

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'],

)

history = model.fit(

np.array(X\_train),

to\_categorical(Y\_train),

epochs=40,

batch\_size=32,

validation\_data=(np.array(X\_test), to\_categorical(Y\_test))

)

# Проверка точности предсказания нейронной сети

\_, score\_test = model.evaluate(np.array(X\_test), to\_categorical(Y\_test), verbose=1)

\_, score\_train = model.evaluate(np.array(X\_train), to\_categorical(Y\_train), verbose=1)

print('Train: %.3f, Test: %.3f' % (score\_train, score\_test))

#%%

#построение графика размера ошибки модели в процессе обучения

plt.subplot(211)

plt.title('Loss')

plt.plot(history.history['loss'], label='train')

plt.plot(history.history['val\_loss'], label='test')

plt.legend()

#построение графика точности модели в процессе обучения

plt.subplot(212)

plt.title('Accuracy')

plt.plot(history.history['acc'], label='train')

plt.plot(history.history['val\_acc'], label='test')

plt.legend()

plt.show()

#%%

stats = pd.DataFrame(history.history)

plotly\_df(df = stats[["loss", "val\_loss"]], path = filepath + "Данные/Графики/", title = "Сокращение ошибки в процессе обучения нейросети. Определение кол-ва людей в комнате", names = ['train', 'test'])

plotly\_df(df = stats[["acc", "val\_acc"]], path = filepath + "Данные/Графики/", title = "Повышение точности в процессе обучения нейросети. Определение кол-ва людей в комнате", names = ['train', 'test'])

#%%

#Сохранение весов модели

model.save\_weights('model\_occupancy.h5')

#%%

#Загрузка модели

#model = Sequential([

# Dense(64, activation='relu', input\_shape=(8,)),

# Dense(64, activation='relu'),

# Dense(3, activation='softmax'),

#])

#Загрузка сохраненных ранее весов

#model.load\_weights('model.h5')

#Предсказание класса для пяти строк тестовой выборки

predictions = model.predict(np.array(X\_test)[2:7])

print(np.argmax(predictions, axis=1)) #[2 1 2 2 0]

#Сравнение с истинными значениями

print(np.array(Y\_test[:5])) #[2. 1. 2. 2. 0.]

#%%

#Сохранение графа нейронной сети

from keras.utils import plot\_model

plot\_model(model, to\_file= filepath + 'Данные/Графики/model1.png', show\_shapes=True)

#%%

#ОПРЕДЕЛЯЕМ ВИД ДЕЯТЕЛЬНОСТИ

#Делим ровно выборки

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

X\_scaled, Y["act"], test\_size=0.33, random\_state=42, stratify = Y)

#%%

from keras.models import Sequential

from keras.layers import Dense

model = Sequential([

Dense(64, activation='relu', input\_shape=(9,)),

Dense(64, activation='relu'),

Dense(5, activation='softmax'),

])

model.summary()

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'],

)

from keras.utils import to\_categorical

history = model.fit(

np.array(X\_train),

to\_categorical(Y\_train),

epochs=40,

batch\_size=32,

validation\_data=(np.array(X\_test), to\_categorical(Y\_test))

)

#Определение точности прогноза

\_, score\_test = model.evaluate(np.array(X\_test), to\_categorical(Y\_test), verbose=1)

\_, score\_train = model.evaluate(np.array(X\_train), to\_categorical(Y\_train), verbose=1)

print('Train: %.3f, Test: %.3f' % (score\_train, score\_test))

#%%

stats = pd.DataFrame(history.history)

#построение графика размера ошибки модели в процессе обучения

plt.subplot(211)

plt.title('Loss')

plt.plot(stats['loss'], label='train')

plt.plot(stats['val\_loss'], label='test')

plt.legend()

#построение графика точности модели в процессе обучения

plt.subplot(212)

plt.title('Accuracy')

plt.plot(stats['acc'], label='train')

plt.plot(stats['val\_acc'], label='test')

plt.legend()

plt.show()

#%%

stats = pd.DataFrame(history.history)

plotly\_df(df = stats[["loss", "val\_loss"]], path = filepath + "Данные/Графики/", title = "Сокращение ошибки в процессе обучения нейросети. Определение вида деятельности", names = ["train", "test"])

plotly\_df(df = stats[["acc", "val\_acc"]], path = filepath + "Данные/Графики/", title = "Повышение точности в процессе обучения нейросети. Определение вида деятельности", names = ['train', 'test'])

#%%

#Сохранение весов модели

model.save\_weights('model\_activity.h5')

#%%

#Загрузка модели

#model = Sequential([

# Dense(64, activation='relu', input\_shape=(8,)),

# Dense(64, activation='relu'),

# Dense(3, activation='softmax'),

#])

#Загрузка сохраненных ранее весов

#model.load\_weights('model.h5')

#Предсказание класса для первых пяти строк тестовой выборки

predictions = model.predict(np.array(X\_test)[:5])

print(np.argmax(predictions, axis=1)) #[4 2 1 2 0]

#Сравнение с истинными значениями

print(np.array(Y\_test[:5])) #[4. 2. 1. 2. 0.]

#%%

#Отрисовка архитектуры

plot\_model(model, to\_file= filepath + 'Данные/Графики/model2.png', show\_shapes=True)

#%%

## Приложение B. Техническое задание

**Техническое задание на проект**

**студента образовательной программы «Информационная аналитика в управлении предприятием» группы ИАУП-18-1**

Негановой Эльвиры Анатольевны

**Цель проекта:** улучшение качества управления системой Умного офиса.

**Заказчик проекта:** кафедра информационных технологий в бизнесе, группа исследования киберфизических систем.

**Сроки реализации проекта:** отчет о проделанной работе передается руководителю проекта по окончанию срока реализации проекта. Плановые сроки начала проекта: апрель 2019 года. Плановые сроки окончания проекта: июнь 2019 года.

**Требования к результату проекта:** реализованная нейронная сеть, способная обрабатывать сигналы, поступающие от датчиков.

**Требования к исполнителю:** исполнителем является студент магистратуры НИУ ВШЭ Пермь по программе “Информационная аналитика в управлении предприятием” Неганова Эльвира Анатольевна. К моменту выполнения проекта студентом должны быть освоены такие дисциплины, как “Архитектура информационных систем управления”, “Научно-исследовательский семинар”, “Системный анализ и проектирование”, “Совершенствование архитектуры предприятия”, “Средства Business Intelligence и системы поддержки принятия решений”, “Экономико-математическое моделирование”.

Исполнитель обязан ответственно и самостоятельно подойти к выполнению поставленной задачи, полно и в указанные сроки предоставить результаты выполнения проекта. Исполнитель совместно с Заказчиком должны разработать техническое задания проекта, уточнить план-график и основные этапы выполнения проекта, определить промежуточные результаты и процедуры контроля и оценки. Участник проекта обязан подготовить отчетные материалыо своей проектной работе.

**График реализации проекта**:

**Таблица 1.1. График реализации проекта**

| № | Дата | Результат |
| --- | --- | --- |
| 1 | 21.04.19 | Сформулированы требования к результату проекта, оформлено Техническое задание |
| 2 | 28.04.19 | Выполнен сравнительный анализ аналогов |
| 3 | 15.05.19 | Выполнен анализ инструментов разработки |
| 4 | 20.05.19 | Выполнено проектирование |
| 5 | 10.06.19 | Выполнена разработка прототипа приложения (изделия) |
| 6 | 16.06.19 | Написан отчет по проекту и подготовлена презентация для защиты проекта на кафедре |
| 7 | 17.06-22.06 | Защита на кафедре |

**Критерии завершенности проекта:**

Проект считается завершенным, если выполнены следующие условия:

1. Аналитическая работа проведена в полном объеме, в т.ч. проведен анализ и обоснован выбор способа построения и типа нейронной сети,
2. Выполнено построение и обучение нейронной сети.
3. Построенная нейронная сеть соответствует критериям качества, указанным в Техническом задании.
4. Разработанная программа реализует все заявленные функциональные требования.
5. Работа программы выполняется без ошибок, тестирование работы программы завершено успешно.
6. Исполнителем подготовлена и представлена на проверку отчетность в полном составе.
7. Результат выполнения проекта представлен заказчику и принят им, работа исполнителем над проектом оценена по 10-ти балльной шкале.

**Критерии качества продукта/результата:**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Критерий** | **Подтверждение** | **Баллы** |
| 1 | Точность предсказания | не менее 90% |  |

**Форма отчетности:**

1. Проектная заявка.
2. Техническое задание на проект.
3. Отчет о выполнении проекта.
4. Программа, реализующая нейронную сеть.

**Трудоемкость проекта в зачетных единицах**: 2 з.е.

**Руководитель проекта** доцент кафедры информационных технологий в бизнесе, к.т.н., доцент, Викентьева Ольга Леонидовна.

Студент факультета экономики,

менеджмента и бизнес-информатики Неганова Эльвира Анатольевна