МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(Национальный исследовательский университет)
Политехнический институт. Энергетический факультет
Кафедра «Электрические станции, сети и системы электроснабжения»

РАБОТА ПРОВЕРЕНА
Рецензент, _____
(должность)
_____ / _____ /
(подпись и печать) (И.О. Фамилия)
« 9 » июня 2020 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____ / И.М. Кирпичникова /
« 9 » июня 2020 г.

«Выбор архитектуры и параметров нейросети к задаче краткосрочного прогнозирования электропотребления в узлах электрической сети»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 13.03.02.2020.154 ВКР

Руководитель, доцент
_____ / В.С. Павлюков /
« 11 » июня 2020 г.

Автор работы
студент группы П-471
_____ / Д.С. Подберезко /
« 11 » июня 2020 г.

Нормоконтролёр, доцент
_____ / В.С. Павлюков /
« 11 » июня 2020 г.

Челябинск 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(Национальный исследовательский университет)
Политехнический институт. Энергетический факультет
Кафедра «Электрические станции, сети и системы электроснабжения»
Направление 13.03.02 «Электроэнергетика и электротехника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ / И.М. Кирпичникова /
« 4 » марта 2020 г.

ЗАДАНИЕ
на выпускную квалификационную работу бакалавра

Подберезко Денис Сергеевич
Группа П-471

1. Тема выпускной квалификационной работы «Выбор архитектуры и параметров нейросети к задаче краткосрочного прогнозирования электропотребления в узлах электрической сети». утверждена приказом по университету от « ___ » _____ 2020 г. № _____

2. Срок сдачи студентом законченной работы « 12 » мая 2020 г.

3. Исходные данные к работе

Данные потребляемой мощности, метеорологические данные

4. Перечень вопросов, подлежащих рассмотрению

Рассмотреть классические методы прогнозирования, нейронные сети, практическое применение НС.

5. Перечень графического материала

Презентация, выполненная в программе Microsoft PowerPoint.

7. Дата выдачи задания « 14 » января 2020 г.

Руководитель работы __
(подпись)

Задание принял к исполнению _____
(подпись студента)

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов выпускной квалификационной работы	Дата завершения этапа	Отметка руководителя о выполнении
Анализ научно-технической литературы по исследуемому вопросу		
Анализ классических методов прогнозирования		
Анализ интеллектуальных методов прогнозирования		
Изучение гиперпараметров нейронной сети		
Изучение архитектур нейронной сети		
Изучение библиотек Keras и TensorFlow		
Построение различных моделей НС с использованием исходных данных		
Оформление дипломной работы		
Оформление презентации к докладу		

Заведующий кафедрой _____ / И.М. Кирпичникова /

Руководитель работы _____ / В.С. Павлюков/

Студент _____ / Д.С. Подберезко /

АННОТАЦИЯ

Актуальность. Прогнозирование потребления электрической энергии на сегодняшний день является актуальной задачей электросетевых компаний и генерирующих станций. Это связано с высоким уровнем развития электрических сетей, появлением всё большего количества потребителей, а также необходимостью соблюдения баланса потребления и выработки электроэнергии.

Цель работы – исследование методов прогнозирования электрических нагрузок, которые позволят с высокой точностью предсказывать мощность потребляемой энергии в рассматриваемом узле энергосистемы.

Задачи. Для достижения поставленной цели ставятся следующие задачи:

• произвести обзор научно-технической литературы, связанной с данной областью исследования;

• систематизировать полученную базу ретроспективных данных электрической нагрузки для промышленного предприятия, а именно:

- потребляемая почасовая мощность;

- значения температуры окружающего воздуха;

- значения влажности воздуха;

- атмосферное давление;

- и другие параметры.

• произвести анализ классических и интеллектуальных методов прогнозирования;

• разработать модель краткосрочного прогнозирования электрической нагрузки;

• показать эффективность применения ИНС.

Практическая значимость. Практическая значимость заключается в том, что разработанная модель при интеграции в готовый продукт (программу для ЭВМ) может быть использована для управления режимами в Челябинском РДУ, Свердловском РДУ, т.е. во всех филиалах АО «СО ЕЭС».

Структура и объём. Работа состоит из введения, трёх глав, заключения, библиографического списка из 7 наименований, приложения. Основной материал содержит 49 илл. Общий объем работы составляет 55 стр.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В электрических сетях важной задачей является соблюдение баланса мощностей потребляемой и вырабатываемой электроэнергии. Так при несоблюдении баланса активной мощности меняется частота сети (нормально допустимое отклонение частоты сети $\pm$ 0,2 Гц). При несоблюдении баланса реактивной мощности меняется напряжение сети (нормальное допустимое отклонение напряжения $\pm$ 5 % от номинального напряжения сети). Поэтому необходимо уметь прогнозировать спрос на электроэнергию в любой части энергосистемы с большой точностью.

Используемые методы прогнозирования имеют очень большой недостаток – низкую точность и не малые трудозатраты. Но с развитием вычислительных мощностей GPU и CPU компьютеров, а также облачных вычислений, стало возможным практическое применение нейронных сетей в прикладных задачах.

1 КЛАССИЧЕСКИЕ МЕТОДЫ ПРОГНОЗИРОВАНИЯ

Следует отметить, что классические методы прогнозирования подразделяются в зависимости от периода прогнозирования на:

– краткосрочный (от часа до суток)

– среднесрочный (от недели до года)

– долгосрочный (более года)

Краткосрочное и среднесрочное прогнозирование имеет общий характер построения моделей, поэтому в работе также косвенно будет затрагиваться вопрос среднесрочного прогнозирования. Долгосрочное прогнозирование имеет совершенно другую логику построения моделей и способов для прогнозирования, требующее более глубокий анализ каждого отдельно взятого объекта.

Существующие методы прогнозирования подразделяются на классические и интеллектуальные.

1.1 Классические методы прогнозирования

Классические методы, в свою очередь, подразделяются на статистические и аналитические.

Статистические методы – основаны на теории вероятности и сборе статистических данных.

Аналитические методы – основаны на определенных выявленных закономерностях, например, гармонический ряд, математическая модель Карла Иберла «Факторный анализ», способ Сааренда).

Недостатками этих методов являются:

- большая погрешность (до 10-20%, которая не укладывается в рамки ГОСТа качества электроэнергии);
- сложность и трудоёмкость расчётов, требующих набор высококвалифицированных специалистов;
- Линейность.

1.2 Интеллектуальные методы прогнозирования

Интеллектуальными методы прогнозирования строятся на определённых алгоритмах, характерной чертой которых является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач. Этими вопросами занимается раздел науки о данных – машинное обучение. Наиболее распространенными считаются следующие алгоритмы:

- Дерево принятия решений – это метод поддержки принятия решений, основанный на использовании древовидного графа: модели принятия решений, которая учитывает их потенциальные последствия (с расчётом вероятности наступления того или иного события), эффективность, ресурсозатратность [1].
- Байесовская классификация – наивные байесовские классификаторы относятся к семейству простых вероятностных классификаторов и берут начало из теоремы Байеса, которая применительно к данному случаю рассматривает функции как независимые (это называется строгим, или наивным, предположением) [1].
- Метод наименьших квадратов – один из вариантов реализации линейно регрессии [1].
- Логистическая регрессия – это способ определения зависимости между переменными, одна из которых категориально зависима, а другие независимы. Для этого применяется логистическая функция (аккумулятивное логистическое распределение) [1].
- Нейронные сети [1].
- Другие методы, например, метод K-ближайших соседей (KNN), Бэггинг и случайный лес, Бустинг [1].

2 ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

2.1 Определение нейронной сети и нейрона

С появлением быстродействующих вычислительных устройств (начало 20 века) искусственные нейронные сети (ИНС) получили широкую известность, так как мощности ЭВМ стали достаточно велики для работы с ними.

Нейронные сети являются подразделом машинного обучения, которые обладают более высокой точностью прогнозирования, гибкостью отстройки для определенной задачи, а также являются стремительно развивающейся отраслью машинного обучения, в следствии чего имеют наибольшее распространение и поддержку среди пользователей, а также больше возможностей для их реализации и применения.

Нейрон – элемент, который выполняет следующие функции:

- Прием сигналов от предыдущих элементов сети;
- Комбинирование входных сигналов;
- Вычисление выходного сигнала;
- Передача выходного сигнала следующим элементам нейронной сети.

Рисунок 1 – Биологический нейрон, соединенный синапсами с другими нейронами

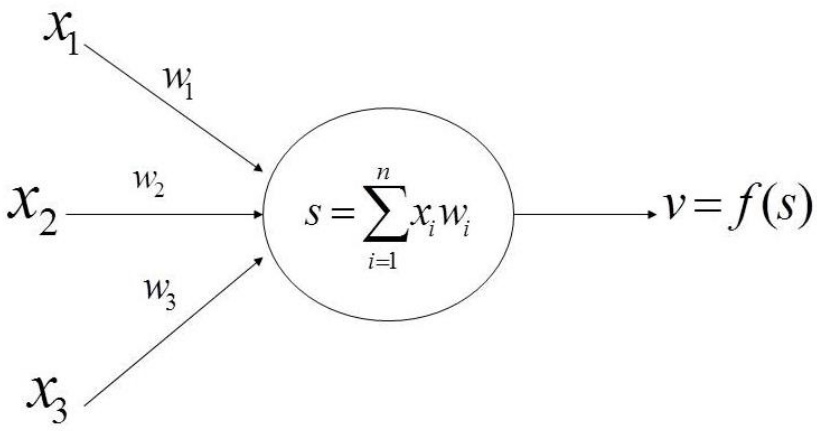


Рисунок 2 – Схема искусственного нейрона, (x_1, x_2, x_3, … x_n – входные сигналы, w_1, w_2, w_3, … w_n– весовые коэффициенты, $f(s)$ – функция активации, v – выходной сигнал)

Искусственная нейронная сеть – объединенные в общую структуры нейроны, соединенные определенным образом, в зависимости от архитектуры данной сети.

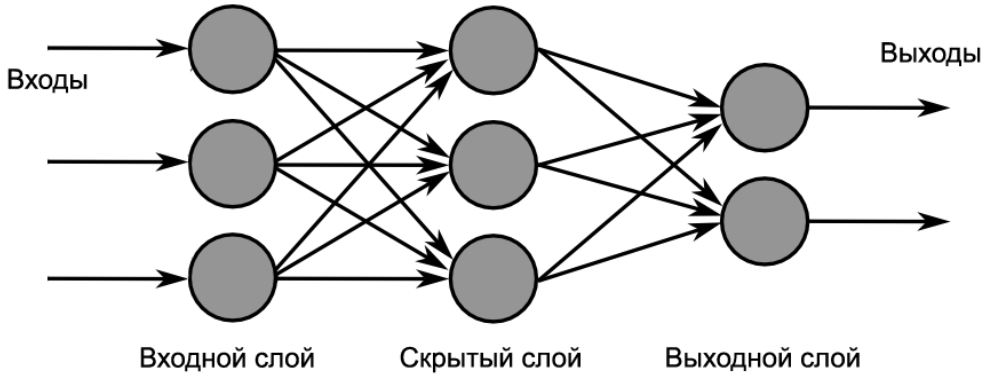


Рисунок 3 – Классическая структура искусственной нейронной сети

2.2 Основные гиперпараметры нейронной сети

2.2.1 Весовые коэффициенты

Вес связи – значение, которое усиливает или ослабляет сигнал, который передается по данной связи. Он характеризует «важность» каждого входа, влияющую на общее решение нейрона.

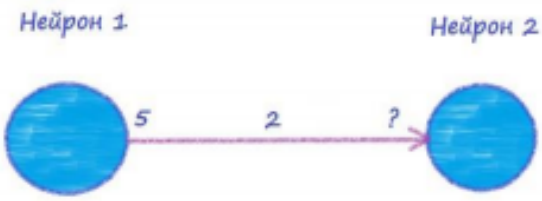


Рисунок 4 – Передача выходного сигнала следующему элементу нейронной сети

На примере рисунок 4 можем видеть, что входной сигнал нейрона 2 будет 5*2=10.

Как видно из рисунок 5, сигнал может быть не один, тогда на входе мы получаем следующее:

$$net_j = \sum_{i=1}^{N} x_i \cdot w_{ij},$$

где net_j – результат суммирования (комбинирования) всех входных сигналов нейрона j (взвешенная сумма); N – количество элементов, предающих свои входные сигналы на вход сигнала j; w_{ij} – вес связи, соединяющий нейрон i с нейроном j.

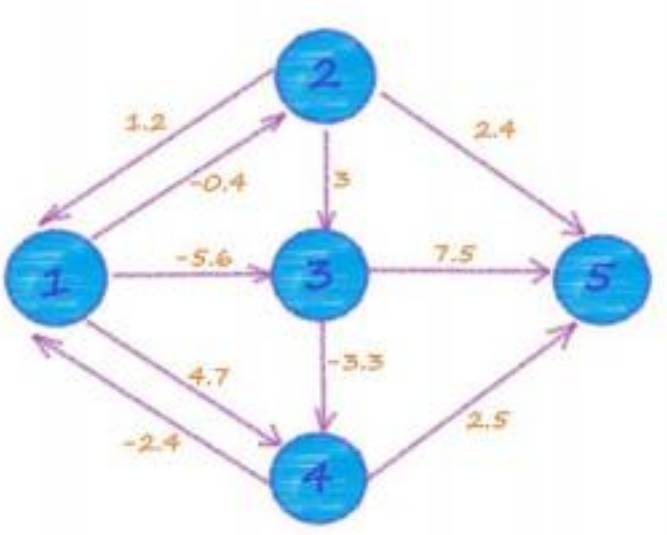


Рисунок 5 – Пример нейронной сети

Для представления связей используют матричную модель W.

$$W = \begin{bmatrix} 0 & -0,4 & -5,6 & 4,7 & 0 \\ 1,2 & 0 & 3 & 0 & 2,4 \\ 0 & 0 & 0 & -3,3 & 7,5 \\ -2,4 & 0 & 0 & 0 & 2,5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.2.2 Функции активации

После нахождения взвешенной суммы нейрон должен обработать ее и сформировать выходной сигнал. Для этого и нужна функция активации.

Таким образом, **функция активации** – функция, преобразующая взвешенную сумму (net_j) в число, которое и будет выходом нейрона.

На сегодняшний день можно встретить самые разные функции активации, которые обозначают $\phi(net)$. Результат на выходе нейрона и есть значение этой функции ($out = \phi(net)$).

Рассмотрим самые распространенные функции активации:

2.2.2.1 Пороговая функция (Функция Хевисайда)

Является самым простым видом функции. Значение на выходе может быть либо 0, либо 1. Если значение входного сигнала $x = net$ ниже определенного порога (принимается за 0 на рисунок 6), то значение на выходе, соответственно, равно 0. Если больше, то 1.

Графическое изображение функции представлено на рисунок 6.

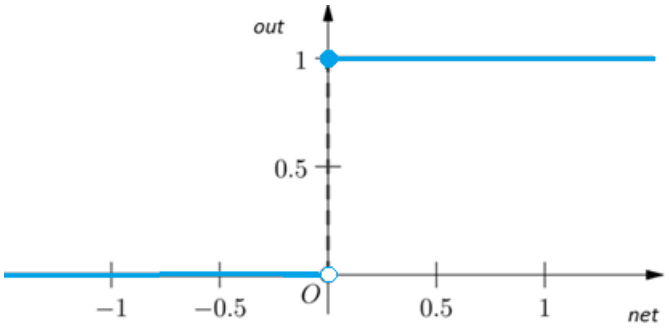


Рисунок 6 – Графическое изображение функции Хевисайда

Математическая интерпретация функции:

$$out = \begin{cases} 0, & \text{если } net < 0, \\ 1, & \text{если } net \geq 0. \end{cases}$$

Данная функция практически не применяется в нейронных сетях на данный момент.

2.2.2.2 Линейная функция

Представляет собой прямую, пропорциональную взвешенной сумме.

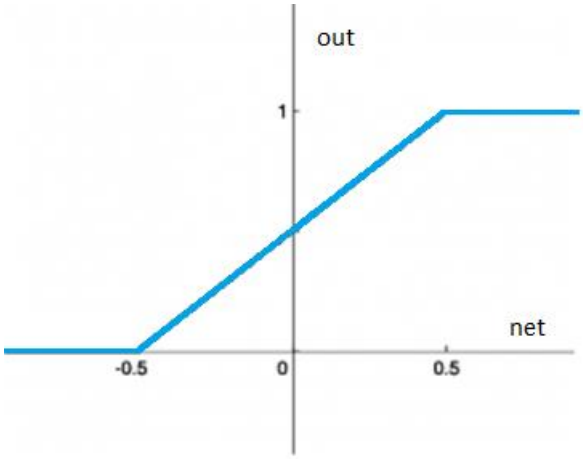


Рисунок 7 – Линейная функция

Данная функция позволяет получить спектр значений, а не бинарный ответ. Однако достоинство функции перекрывают ее недостатки:

- Производная является постоянной, следовательно, градиент тоже постоянный, и спуск производится по постоянному градиенту. Это говорит о том, что вследствие ошибочных предсказаний изменения, сделанные обратным распространением ошибки, постоянны и не зависят от изменения на входе;
- Сочетание линейных функций активации является результатом другой линейной функции, т.е. нейронная сеть (НС) подобна одному слою.

Данная функция используется в задаче регрессии на выходном слое, который состоит из одного нейрона, чтобы принять значения с нейронов предыдущего слоя и выдать их сумму в виде результата работы нейронной сети.

2.2.2.3 Сигмоидальная функция

Сигмоидальная функция является одной и самых распространённых функций для нейронных сетей, её возможно применять в любых видах задач.

Существует семейство таких функций, применяемых в ИНС. Аналитически записываются они так:

$$out = \frac{1}{1 + \exp(-a \cdot net)}.$$

Параметр a показывает степень крутизны функции, что можно увидеть на Рисунок 8.

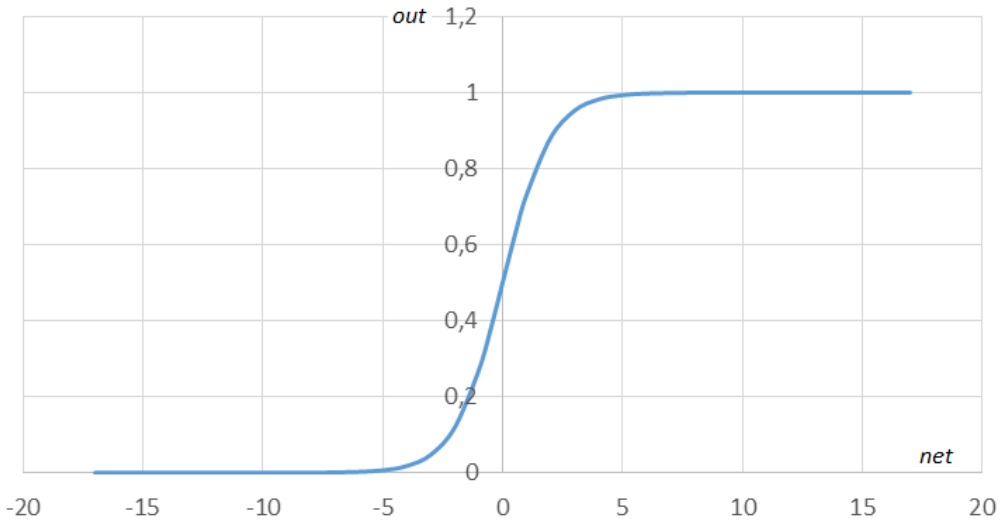


Рисунок 8 – Сигмоидальная функция

К преимуществам данной функции можно отнести следущие свойства:

- Значение этой функции может иметь любое значение от 0 до 1, в зависимости от входного сигнала и параметра a;
- Параметром a можно регулировать крутизну функции;
- Во всех точках имеет производную, которая выражается через сигмоидальную функцию:

$$\frac{d(out(net))}{d(net)} = out(net) \cdot (1 - out(net)).$$

Недостатком данной функции является:

- Затухание градиента в глубоких нейронных сетях;
- Относительно большая ресурсоёмкость вычислений.

2.2.2.4 Гиперболический тангенс

Функция записывается следующим образом:

$$out = tanh\left(\frac{net}{a}\right).$$

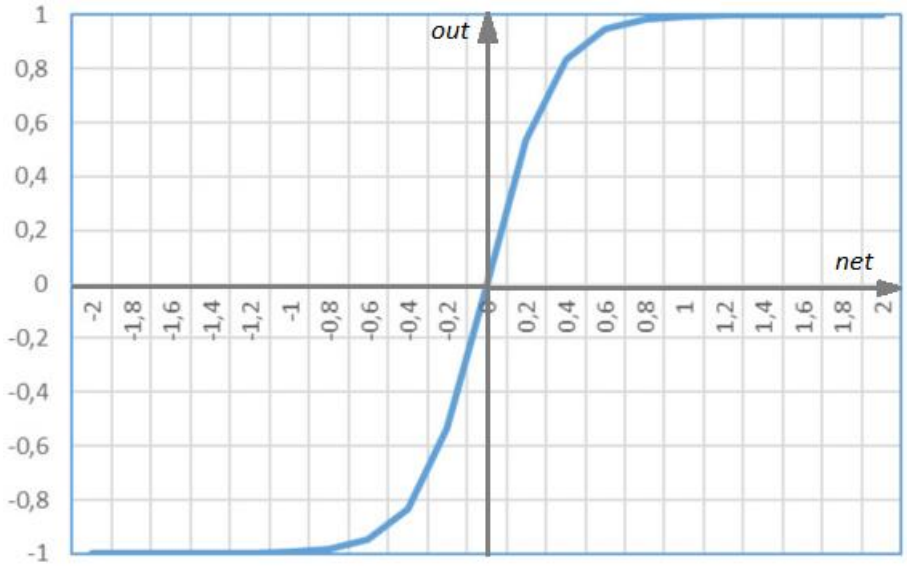


Рисунок 9 – Гиперболический тангенс

Достоинства:

- Данная функция имеет сходство с сигмоидой, однако её значения находятся в пределах от -1 до 1, что позволяет применять её, когда нужны отрицательные значения на выходе нейрона;
- Простое вычисление производной, через саму же функцию:

$$\frac{d(out(net))}{d(net)} = 1 - out(net)^2.$$

Недостатки:

- Затухание или увеличение градиента,
- Достаточно большая ресурсоёмкость вычислений.

2.2.2.5 ReLU

ReLU (rectified linear unit) – выпрямленная линейная единица.

Аналитическое представление функции:

$$out(net) = \max(0, net).$$

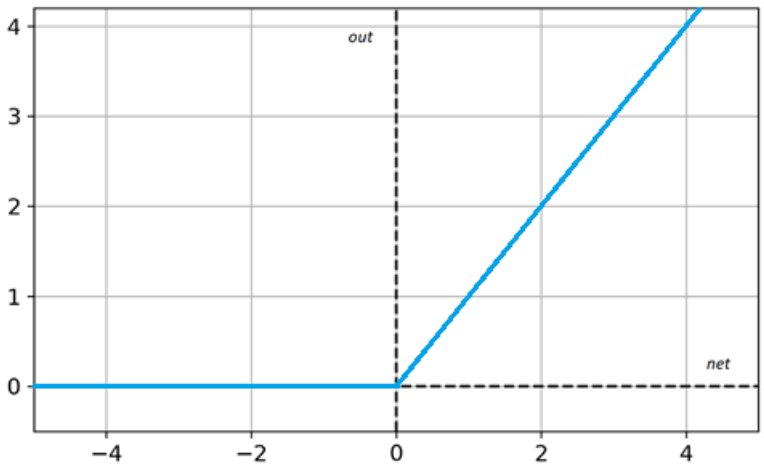


Рисунок 10 – ReLU

Преимущества использования ReLU:

- Её производная равна либо единице, либо нулю, и поэтому не может произойти разрастания или затухания градиентов, так как умножив единицу на дельту ошибки мы получим дельту ошибки, если же бы использовали другую функцию, например, гиперболический тангенс, то дельта ошибки могла, либо уменьшиться, либо возрасти, либо остаться такой же, то есть, производная гиперболического тангенса возвращает число с разным знаком и величиной, что можно сильно повлиять на затухание или разрастание градиента. Более того, использование данной функции приводит к прореживанию весов [2];
- вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как ReLU может быть реализован с помощью простого порогового преобразования матрицы активаций в нуле [2].

Из недостатков можно отметить, что ReLU не всегда достаточно надежна и в процессе обучения может выходить из строя («умирать»). Например, большой градиент, проходящий через ReLU, может привести к такому обновлению весов, что данный нейрон никогда больше не активируется. Если это произойдет, то, начиная с данного момента, градиент, проходящий через этот

нейрон, всегда будет равен нулю. Соответственно, данный нейрон будет необратимо выведен из строя. Например, при слишком большой скорости обучения (learning rate), может оказаться, что до 40% ReLU «мертвы» (то есть, никогда не активируются). Эта проблема решается посредством выбора надлежащей скорости обучения [2].

2.2.2.5 ELU

Эта функция похожа на ReLU, но включает в себя экспоненту, что делает её вычисление более затруднительным. Но это даёт следующие преимущества:

- При отрицательных входных данных градиент не будет нулевым;
- Её можно использовать, когда входные данные сильно зашумлены

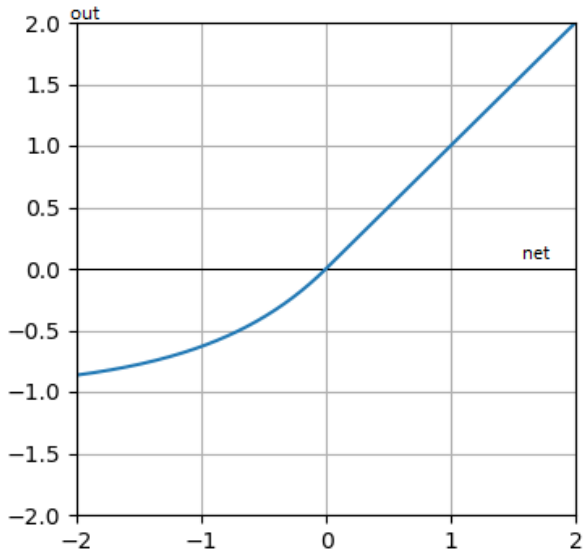


Рисунок 11 – ELU

2.2.3 Глубина архитектуры ИНС

2.2.3.1 Однослойные нейронные сети

В однослойных нейронных сетях входные данные попадают на входной слой (input layer) и после преобразования сразу поступают на выходной слой (output layer).

Однослойная НС представлена на рисунок 12

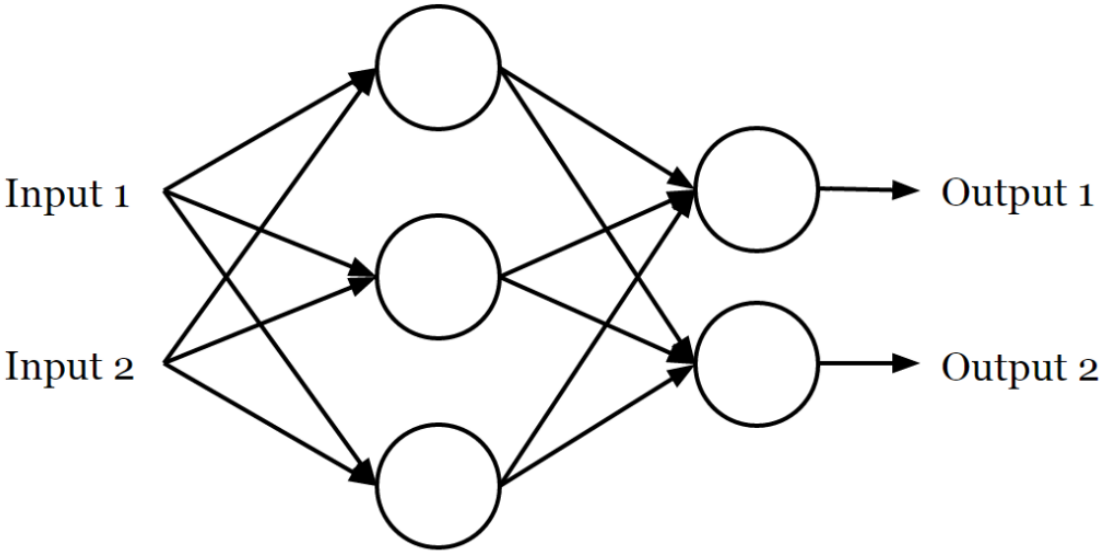


Рисунок 12 – Однослойная НС

Входной слой (сенсорный), как правило, не считается слоем, так как на него, непосредственно, поступают входные данные и поэтому сеть называется однослойной.

Данные сети имеют следующие преимущества:

- Не затухающий или не разрастающийся градиент, если используются сигмоидальные и подобные ей функции активации;
- Быстрая скорость обучения.

2.2.3.2 Многослойные нейронные сети (глубокие нейронные сети)

Данный вид НС состоит их входного слоя, скрытых и выходного (см. Рисунок 3).

Достоинства многослойных нейронных сетей:

- Имеют большую гибкость при настройке НС
- Как правило, выдают лучший результат
- Применимы во всех прикладных задачах (от задач регрессии до компьютерного зрения)

Недостатки:

- Большая ресурсозатратность и сложность вычислений
- Затухающий или разрастающийся градиент, при использовании сигмоидальной и подобной ей функций активации.

2.2.4 Распространенные алгоритмы нахождения ошибки (оптимизаторы)

2.2.4.1 Алгоритм градиентного спуска

Алгоритм градиентного спуска — алгоритм нахождения локального экстремума (минимума или максимума) функции с помощью движения вдоль градиента.

Рассмотрим градиентный спуск на примере, когда признак всего один (парная регрессия)

Модель:

$$a(x) = w_1 \cdot x + w_0.$$

Два параметра:

$$w_1 \text{ и } w_0.$$

Функционал:

$$Q(w_0, w_1, X) = \frac{1}{l}\sum_{i=1}^{l}(w_1 \cdot x_i + w_0 - y_i)^2,$$

где Q – Функционал ошибки (функция потерь), вычисляемая по методу среднеквадратичной ошибки;

x_i– значение признака;

w_1 – весовой коэффициент;

w_0 – свободный коэффициент (сдвиг);

y_i – истинное значение функции.

Алгоритм:

1) Инициализируется вектор весов:

$$w^0 = 0.$$

2) До сходимости повторяется градиентный шаг:

Цикл по t = 1, 2, 3, …:

$$w^t = w^{t-1} - \eta_t \nabla Q(w^{t-1}, X).$$

Если $\| w^t - w^{t-1} \| < \varepsilon$, то завершить,

где η_t – шаг градиентного спуска.

Чтобы вычислить градиентный спуск, нужно вычислить градиент функционала ошибки по всем его параметрам, т.е. найти его частные производные:

$$\nabla Q(w^{t-1}, X);$$

$$\frac{\partial Q}{\partial w_1} = \frac{2}{l}\sum_{i=1}^{l}(w_1 \cdot x_i + w_0 - y_i) \cdot x_i;$$

$$\frac{\partial Q}{\partial w_0} = \frac{2}{l}\sum_{i=1}^{l}(w_1 \cdot x_i + w_0 - y_i).$$

Тогда значение ошибки в зависимости от количества итераций будет монотонно уменьшаться до асимптоты, представленной на рисунок 13:

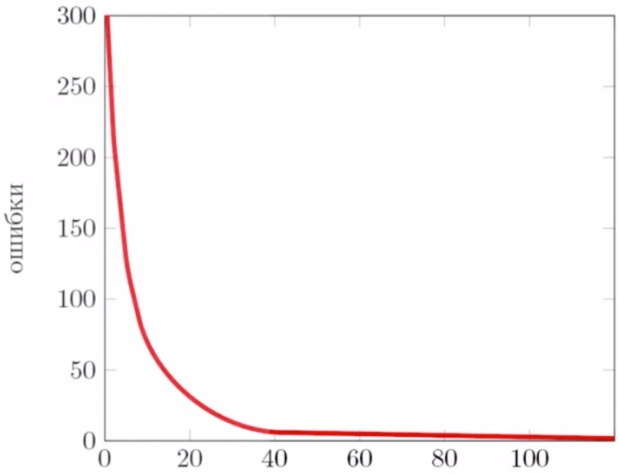


Рисунок 13 – Значение ошибки

Размера шага сильно влияет на нахождения экстремума, что показано на рисунок 14:

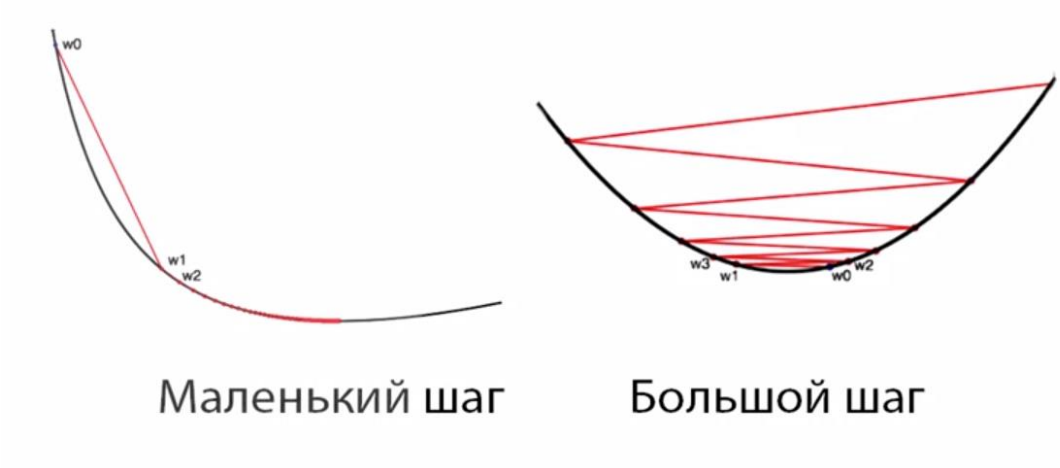


Рисунок 14 – Влияние величины шага η_t на сходимость к минимум

Решением данной проблемы может быть уменьшение шага в зависимости от количества итераций, т.е. сделать его переменным.

Для многомерной регрессии, когда количество признаков больше одного, подход для вычисления градиента будет аналогичным, однако функционал ошибки и градиент будут вычисляться по формулам (1-2):

$$Q(w, X) = \frac{1}{l} \| X \cdot w - y \|^2 \rightarrow min, \quad (1)$$

$$\nabla_w Q(w, X) = \frac{2}{l} X^T (X \cdot w - y). \quad (2)$$

2.2.4.2 Стохастический градиентный спуск (SGD)

Стохастический градиентный спуск – это модификация градиентного спуска, которая особенного хорошо подходит для обучения линейных моделей.

Распишем формулу (2) для j-ой компоненты вектора градиента, получим формулу (3):

$$\frac{\partial Q}{\partial w_j} = \frac{2}{l} \sum_{i=1}^{l} x_i^j (\langle w, x_i \rangle - y_i), \quad (3)$$

где *l* – длина обучающей выборки.

Если выборка очень большая, то один градиентный шаг будет занимать слишком много времени, что является недостатком классического градиентного спуска.

Рассмотрим алгоритм стохастического градиентного спуска:

1. Инициализируем вектор весов:

$$w^0 = 0.$$

2. В отличии от классического градиентного спуска на одной итерации мы вычитаем не вектор градиента, вычисленный по всей выборке, а случайно выбираем один объект x_i из обучающей выборки и дальше вычисляем градиент функционала только на этом объекте, т.е. градиент только одного слагаемого в функционале ошибки и вычитаем именно этот градиент из текущего приближения весов.

Цикл по t = 1, 2, 3, …:

выбирается случайный объект x_i из X

$$w^t = w^{t-1} - \eta_t \nabla Q(w, \{x_i\}).$$

Если $\| w^t - w^{t-1} \| < \varepsilon$, то завершить цикл.

У SGD график сходимости уже будет иметь пилообразный вид, так как ошибка уменьшается не на всей выборке, а происходит её уменьшение только на одном объекте на каждой итерации, но при этом ошибка может увеличиться на другом объекте. В целом, ошибка всё равно уменьшается, показано на рисунок 15.

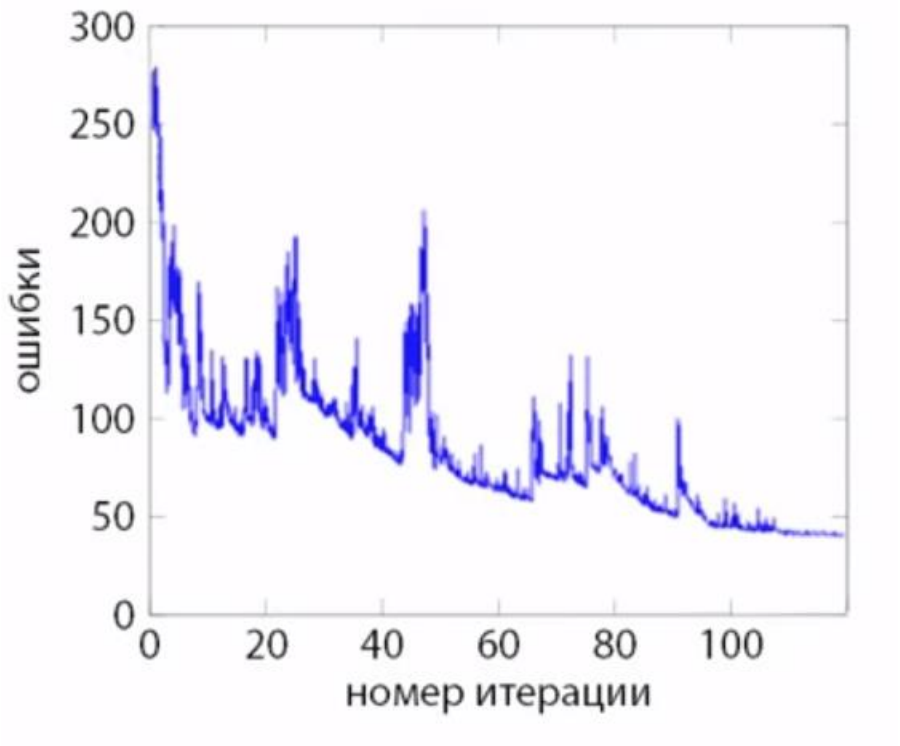


Рисунок 15 – Значение ошибки для SGD

SGD имеет следующие преимущества:

- Быстрее выполняется один шаг;
- Не требует хранения выборки в памяти, т.е. позволяет обучать модель на очень больших выборках, которые не помещаются в памяти;
- Подходит для онлайн-обучения.

2.2.4.3 Adam

Adam − adaptive moment estimation, оптимизационный алгоритм, который сочетает в себе и идею накопления движения, и идею более слабого обновления весов для типичных признаков [3].

В Adam, в отличии от стохастического градиентного спуска, скорость обучения поддерживается для каждого веса сети (параметра) и отдельно адаптируется по мере развития обучения [3].

Adam является популярным алгоритмом в области глубокого обучения, потому что он быстро достигает хороших результатов. Эмпирические результаты показывают, что Adam хорошо работает на практике и выгодно отличается от других методов стохастической оптимизации [3].

Некоторые преимущества использования Adam:

- Просто для реализации;
- Вычислительно эффективен;
- Маленькие требования к памяти;
- Инвариант к изменению масштаба градиента по диагонали;
- Хорошо подходит для задач, которые являются большими с точки зрения данных и / или параметров;
- Подходит для нестационарных целей;
- Подходит для задач с очень шумными / или редкими градиентами;
- Гиперпараметры имеют интуитивно понятную интерпретацию и обычно требуют небольшой настройки [3].

2.2.4.4 RMSprop

Оптимизатор RMSprop (от англ. Root Mean Square Propagation) – это метод, в котором для любого признака возможна настройка скорости обучения. Идея заключается в делении скорости обучения для весов на сгруппированные средние значения недавних градиентов для этого веса. RMSProp показал прекрасную адаптацию скорости обучения в различных приложениях. Метод способен работать с минипакетами, а не только с полными пакетами.

2.2.5 Функции потерь и метрики

В машинном обучении конечная цель заключается в минимизации или максимизации целевой функции. Группа функций, которые минимизируются, называются «функциями потерь». Функция потерь используется как показатель эффективности модели прогнозирования с точки зрения возможности прогнозирования ожидаемого результата.

Существует несколько видов потерь регрессии.

2.2.5.1 Среднеквадратичная ошибка, квадратичная потеря

Среднеквадратичная ошибка (MSE) – самая используемая функция потери регрессии. Измеряется как среднее квадрата разницы между прогнозами и фактическими наблюдениями. Это касается только средней величины ошибок, независимо от их направления:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n},$$

где y_t − фактическое значение;

y_i^p − прогнозное значение;

n – число наблюдений.

2.2.5.2 Средняя абсолютная ошибка

Еще одна функция потерь, используемая для регрессионных моделей. MAE − это сумма абсолютных различий между целевой и прогнозируемой переменными. Таким образом, MAE измеряет среднюю величину ошибок в наборе прогнозов, не учитывая их направления:

$$MAE = \frac{\sum_{i=1}^{n}\left|y_i - y_i^p\right|}{n}.$$

2.2.5.3 Средняя абсолютная процентная ошибка

Средняя абсолютная процентная ошибка (MAPE) является статистической мерой точности системы прогнозирования. Она измеряет эту точность в процентах:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_t - y_i^p}{y_t}\right|.$$

Средняя абсолютная процентная ошибка проста в понимании и работает лучше всего, если в данных отсутствуют крайности.

2.3 Архитектура нейронных сетей

В настоящий момент времени существует большое количество различных архитектур нейронных сетей. Для дальнейшего анализа рассмотрим наиболее применимые для наших задач.

2.3.1 Нейронная сеть прямого распространения и перцептроны

Передача информации происходит от входа к выходу. Описываются в виде множества слоёв, где каждый слой может состоять из одного или нескольких

нейронов. Нейроны в пределах одного слоя не имеют связей, однако соседние слои обычно полностью связаны. Самая простая нейронная сеть имеет два нейрона во входном слое и один в выходном (рисунок 16), и может использоваться в качестве модели логических вентилей.

Рисунок 16 – Модель нейронной сети прямого распространения

2.3.2 Рекуррентная нейронная сеть (RNN)

Данный класс нейронных сетей хорош для моделирования последовательных данных, таких как временные ряды или естественный язык. В данной модели (Рисунок 17) нейроны получают информацию не только от предыдущего слоя, но и от самих себя предыдущего прохода. Большим недостатком является проблема исчезающего градиента, которая заключается в быстрой потере информации с течением времени.

RNN имеют несколько разновидностей:

- LSTM (Long Short-Term Memory) – в отличии от классических нейронных сетей обладают большей запоминающей способностью, однако имеют более высокие требования к вычислительным мощностям ЭВМ;
- GRU (Gated Recurrent Unit) – обладают относительно высокой запоминающей способностью, однако менее требовательны к вычислительным мощностям ЭВМ.

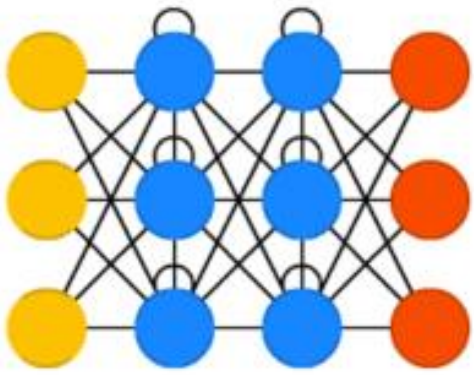


Рисунок 17 – Модель рекуррентной нейронной сети

2.4 Обучение искусственной нейронной сети

Обучение нейронной сети – поиск такого набора весовых коэффициентов, при котором входной сигнал после прохода по сети преобразуется в выходной. Такой подход сопоставляют с биологическими нейросетями, где наш мозг обучается благодаря изменению синапсов – элементов, которые усиливают или ослабляют входной сигнал.

Следует учесть, что для обучения необходимо использовать не один входной сигнал. В противном случае сеть «запомнит правильный ответ». С целью корректного обучения НС создаются обучающие выборки.

Обучающая выборка (Training set) – набор входных сигналов (иногда вместе с правильными выходными сигналами), по которым происходит обучение сети.

Тестовая выборка (Testing set) – конечный набор входных сигналов (иногда вместе с правильными выходными сигналами), по которым происходит оценка качества работы сети.

Рассмотрим способы обучения нейронной сети.

2.4.1 Обучение с учителем

Процесс обучения состоит из следующих этапов:

- Представление сети выборки обучающих примеров;
- Подача образцов на входы и их обработка внутри НС;
- Вычисление выходного сигнала;
- Сравнение с требуемым выходом и вычисление ошибки;
- Изменение весовых коэффициентов связей внутри сети.

Обучение длится до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня. Алгоритм изображен на рисунке 18.

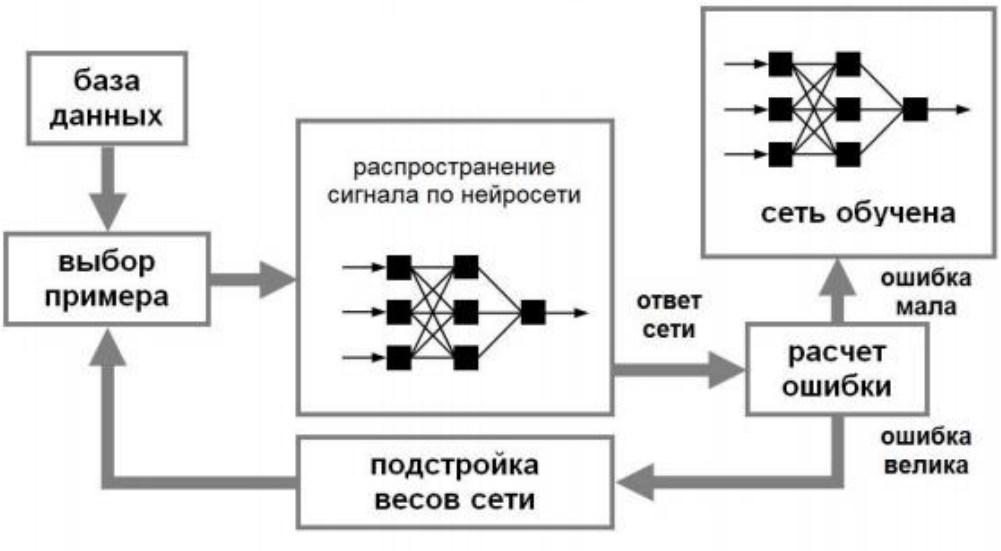


Рисунок 18 – Обучение НС

2.4.2 Обучение без учителя

Отличие заключается в том, что обучающая выборка состоит лишь из входных векторов. Оно применяется тогда, когда нет необходимых правильных ответов.

Обучение состоит из следующих этапов:

- Представление сети выборки обучающих примеров;
- Подача образцов на входы и их обработка внутри НС;
- Вычисление выходного сигнала;
- Кластеризация входных сигналов.

Следовательно, обучение без учителя – вид обучения сети, при котором сеть самостоятельно классифицирует входные сигналы. Правильные (эталонные) выходные сигналы не демонстрируются.

3 ПРИМЕНЕНИЕ НЕЙРОННОЙ СЕТИ ДЛЯ ПРОГНОЗИРОВАНИЯ ПОТРЕБЛЕНИЯ ЭЛЕКТРОЭНЕРГИИ В УЗЛАХ ЭЛЕКТРИЧЕСКОЙ СЕТИ

3.1 Современный подход для прогнозирования нейронных сетей

3.1.1 Средства, с помощью которых можно реализовать обучение нейронной сети

Прогнозирование нейронных сетей можно реализовать в различных программах, таких как:

- Mathcad;
- LabVIEW;
- Matlab.

Но все эти средства имеют один недостаток, слишком медленную скорость обучения, так как они не оптимизированы для данных задач.

3.1.2 Использования средств языка Python

Для реализации нейронной сети был выбран язык программирования Python по нескольким причинам:

1. На сегодняшний день это самый популярный язык программирования, а значит, обладает большим количеством информации по его использованию.
2. Он является высокоуровневым, на нём легко писать код и интерпретировать его.
3. Для Python написано большое количество фреймворков, ориентированных на работу с данными, их обработку, для задач машинного обучения и нейронных сетей.

3.1.4.1 Фреймворки и средства Python для работы с нейронными сетями

В статье [4] проводится анализ существующих Deep Learning технологий и о том, какой фреймворк выбрать, учитывая специфику задачи и способности команды, где библиотеки Torch и Nervana показали наилучшие результаты при тестировании производительности сверточных нейронных сетей. TensorFlow прошел испытания с сопоставимыми результатами, а Caffe и Theano значительно отстали от лидеров. В свою очередь, Microsoft CNTK показал себя наилучшим образом при обучении рекуррентных нейронных сетей (RNN). Авторы другого исследования, сравнивая Theano, Torch и TensorFlow, выбрали

Theano победителем по части обучения RNN. На Рисунок 19 представлены логотипы данных библиотек) [4, 5].

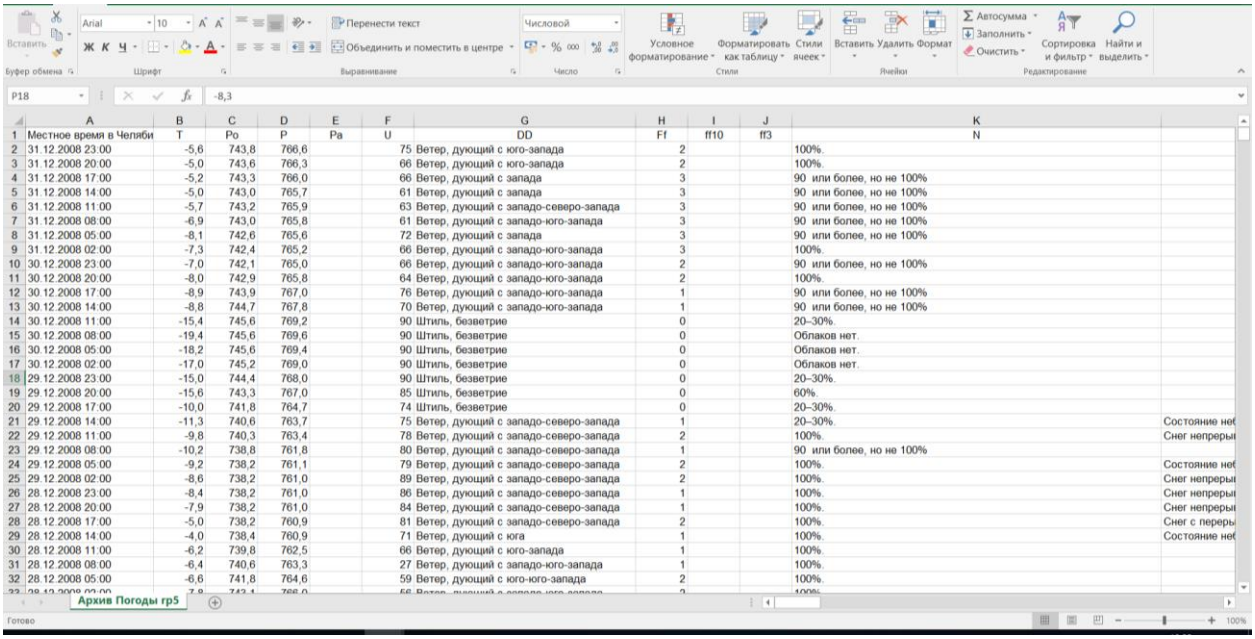
Рисунок 19 – Основные фреймворки

Keras — это высокоуровневая библиотека для быстрой реализации Deep Learning алгоритмов. На данный момент Keras поддерживает два back-end-а, TensorFlow и Theano и в будущем будет получать официальную поддержку от TensorFlow. Более того, в пользу Keras говорит тот факт, что библиотека, по словам ее автора, продолжит выступать в качестве пользовательского интерфейса, который может быть использован с несколькими back-end-ами [4, 5].

Проанализировав достоинства и недостатки различных фреймворков, была выбрана библиотека Keras с back-end-ом – TensorFlow.

3.2 Обработка данных

Исходные данные (рисунок 20) представляются не совсем в удобном виде, присутствует много пустых строк, пропущенных значений и непонятных обозначений. Поэтому нужна тщательная обработка данных (рисунок 21), чтобы нейронная сеть могла без проблем считывать эти данные.



	A	B	C	D	E	F	G	H	I	J	K
1	Местное время в Челяби	T	Po	P	Pa	U	DD	Ff	ff10	ff3	N
2	31.12.2008 23:00	-5,6	743,8	766,6		75	Ветер, дующий с юго-запада	2			100%.
3	31.12.2008 20:00	-5,0	743,6	766,3		66	Ветер, дующий с юго-запада	2			100%.
4	31.12.2008 17:00	-5,2	743,3	766,0		66	Ветер, дующий с запада	3			90 или более, но не 100%
5	31.12.2008 14:00	-5,0	743,0	765,7		61	Ветер, дующий с запада	3			90 или более, но не 100%
6	31.12.2008 11:00	-5,7	743,2	765,9		63	Ветер, дующий с западо-северо-запада	3			90 или более, но не 100%
7	31.12.2008 08:00	-6,9	743,0	765,8		61	Ветер, дующий с западо-юго-запада	3			90 или более, но не 100%
8	31.12.2008 05:00	-8,1	742,6	765,6		72	Ветер, дующий с запада	3			90 или более, но не 100%
9	31.12.2008 02:00	-7,3	742,4	765,2		66	Ветер, дующий с западо-юго-запада	3			100%.
10	30.12.2008 23:00	-7,0	742,1	765,0		66	Ветер, дующий с западо-юго-запада	2			90 или более, но не 100%
11	30.12.2008 20:00	-8,0	742,9	765,8		64	Ветер, дующий с западо-юго-запада	2			100%.
12	30.12.2008 17:00	-8,9	743,9	767,0		76	Ветер, дующий с западо-юго-запада	1			90 или более, но не 100%
13	30.12.2008 14:00	-8,8	744,7	767,8		70	Ветер, дующий с западо-юго-запада	1			90 или более, но не 100%
14	30.12.2008 11:00	-15,4	745,6	769,2		90	Штиль, безветрие	0			20–30%.
15	30.12.2008 08:00	-19,4	745,6	769,6		90	Штиль, безветрие	0			Облаков нет.
16	30.12.2008 05:00	-18,2	745,6	769,4		90	Штиль, безветрие	0			Облаков нет.
17	30.12.2008 02:00	-17,0	745,2	769,0		90	Штиль, безветрие	0			Облаков нет.
18	29.12.2008 23:00	-15,0	744,4	768,0		90	Штиль, безветрие	0			20–30%.
19	29.12.2008 20:00	-15,6	743,3	767,0		85	Штиль, безветрие	0			60%.
20	29.12.2008 17:00	-10,0	741,8	764,7		74	Штиль, безветрие	0			20–30%.
21	29.12.2008 14:00	-11,3	740,6	763,7		75	Ветер, дующий с западо-северо-запада	1			20–30%.
22	29.12.2008 11:00	-9,8	740,3	763,4		78	Ветер, дующий с западо-северо-запада	2			100%.
23	29.12.2008 08:00	-10,2	738,8	761,8		80	Ветер, дующий с западо-северо-запада	1			90 или более, но не 100%
24	29.12.2008 05:00	-9,2	738,2	761,1		79	Ветер, дующий с западо-северо-запада	2			100%.
25	29.12.2008 02:00	-8,6	738,2	761,0		89	Ветер, дующий с западо-северо-запада	2			100%.
26	28.12.2008 23:00	-8,4	738,2	761,0		86	Ветер, дующий с западо-северо-запада	1			100%.
27	28.12.2008 20:00	-7,9	738,2	761,0		84	Ветер, дующий с западо-северо-запада	1			100%.
28	28.12.2008 17:00	-5,0	738,2	760,9		81	Ветер, дующий с западо-северо-запада	2			100%.
29	28.12.2008 14:00	-4,0	738,4	760,9		71	Ветер, дующий с юга	1			100%.
30	28.12.2008 11:00	-6,2	739,8	762,5		66	Ветер, дующий с юго-запада	1			100%.
31	28.12.2008 08:00	-6,4	740,6	763,3		27	Ветер, дующий с западо-юго-запада	1			100%.
32	28.12.2008 05:00	-6,6	741,8	764,6		59	Ветер, дующий с юго-юго-запада	2			100%.

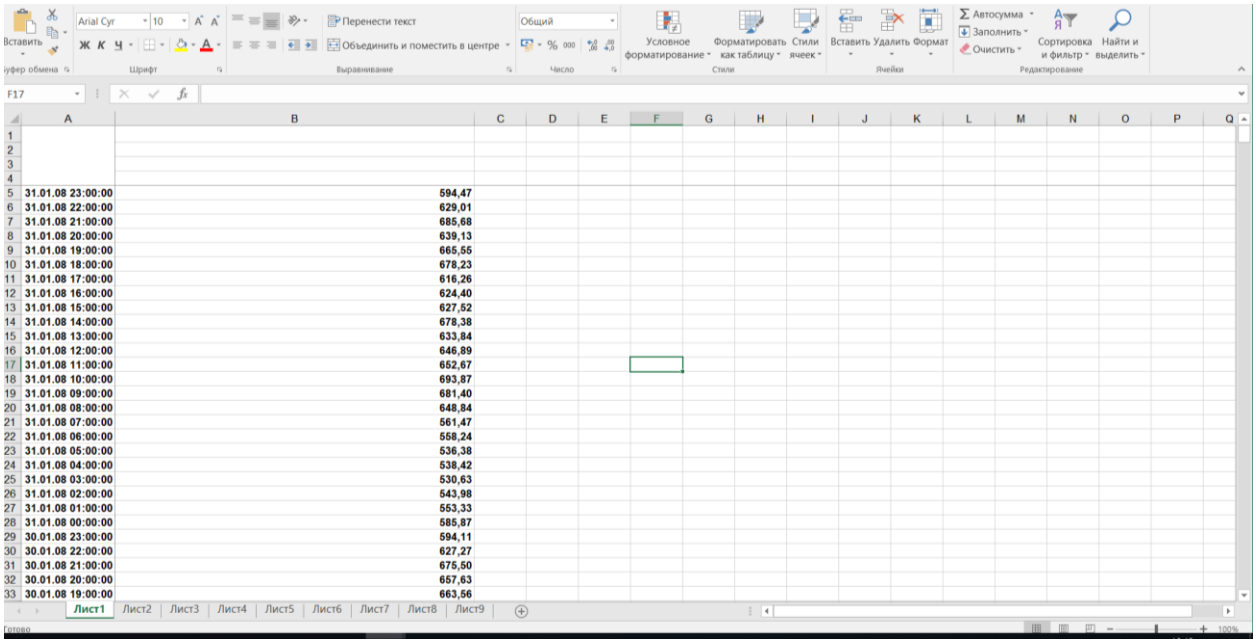
Рисунок 20 – Данные с погодой в районе рассматриваемого узла электронагрузок

Для дальнейшей работы преобразуем данные из файла excel в формат DataFrame (см. Рисунок 21), чтобы с помощью библиотек NumPy и Pandas можно было легко обрабатывать данные и передавать их в нейронную сеть.

	time	T	pressure	air humidity	speed_wind	horizon_visibility
0	31.12.2008 23:00	-5.6	743.8	75.0	2.0	10
1	31.12.2008 20:00	-5.0	743.6	66.0	2.0	10
2	31.12.2008 17:00	-5.2	743.3	66.0	3.0	18
3	31.12.2008 14:00	-5.0	743.0	61.0	3.0	19
4	31.12.2008 11:00	-5.7	743.2	63.0	3.0	19
5	31.12.2008 08:00	-6.9	743.0	61.0	3.0	10
6	31.12.2008 05:00	-8.1	742.6	72.0	3.0	10
7	31.12.2008 02:00	-7.3	742.4	66.0	3.0	10
8	30.12.2008 23:00	-7.0	742.1	66.0	2.0	10
9	30.12.2008 20:00	-8.0	742.9	64.0	2.0	10

Рисунок 21 – Обработанные данные с погодой формата DataFrame

Данные нагрузок были взяты в удобном виде (рисунок 22), но они были разбиты по месяцам в различные файлы, поэтому пришлось их объединить в один набор данных и убрать пустые строки.



	A	B
5	31.01.08 23:00:00	594,47
6	31.01.08 22:00:00	629,01
7	31.01.08 21:00:00	685,68
8	31.01.08 20:00:00	639,13
9	31.01.08 19:00:00	665,55
10	31.01.08 18:00:00	678,23
11	31.01.08 17:00:00	616,26
12	31.01.08 16:00:00	624,40
13	31.01.08 15:00:00	627,52
14	31.01.08 14:00:00	678,38
15	31.01.08 13:00:00	633,84
16	31.01.08 12:00:00	646,89
17	31.01.08 11:00:00	652,67
18	31.01.08 10:00:00	693,87
19	31.01.08 09:00:00	681,40
20	31.01.08 08:00:00	648,84
21	31.01.08 07:00:00	561,47
22	31.01.08 06:00:00	558,24
23	31.01.08 05:00:00	536,38
24	31.01.08 04:00:00	538,42
25	31.01.08 03:00:00	530,63
26	31.01.08 02:00:00	543,98
27	31.01.08 01:00:00	553,33
28	31.01.08 00:00:00	585,87
29	30.01.08 23:00:00	594,11
30	30.01.08 22:00:00	627,27
31	30.01.08 21:00:00	675,50
32	30.01.08 20:00:00	657,63
33	30.01.08 19:00:00	663,56

Рисунок 22 – Первоначальные данные с нагрузкой

Данные с погодными факторами и нагрузкой объединяются и получается конечный набор данных (рисунок 23), на котором уже будет производиться обучение нейронной сети. Обработка данных привела к тому, что от 6790 часов наблюдений изменения нагрузки в течении 2008 года осталось только 2192 из-за удаления строк с отсутствующими значениями хотя бы по одному из признаков, так как нейронные сети не умеют работать с данными типа «NaN» и совмещением с данными погоды, которые снимались с интервалом каждые 3 часа, а не 1 час, как это было с данными нагрузок.

	time	P	T	pressure	air humidity	speed_wind	horizon_visibility	hour	month
0	2008-01-01 02:00:00	509.094910	-6.8	752.8	66.0	0.0	10.0	2	1
1	2008-01-01 05:00:00	469.061890	-10.6	753.3	70.0	0.0	10.0	5	1
2	2008-01-01 08:00:00	456.218994	-10.4	753.3	65.0	0.0	10.0	8	1
3	2008-01-01 11:00:00	466.976196	-8.1	753.3	61.0	2.0	19.0	11	1
4	2008-01-01 14:00:00	452.678894	-7.5	752.6	68.0	2.0	19.0	14	1
5	2008-01-01 17:00:00	507.184998	-7.0	752.3	74.0	0.0	19.0	17	1
6	2008-01-01 20:00:00	536.710815	-7.5	752.1	71.0	1.0	10.0	20	1
7	2008-01-01 23:00:00	531.260620	-7.8	752.0	62.0	3.0	10.0	23	1
8	2008-01-02 02:00:00	469.688293	-17.6	745.9	95.0	2.0	4.0	2	1
9	2008-01-02 05:00:00	479.058777	-19.8	745.9	95.0	0.0	10.0	5	1

Рисунок 23 – Обработанные данные нагрузки, совмещенные с данными погоды, отсортированные по возрастанию

P – нагрузка в рассматриваемом узле (МВт);

T – температура воздуха (градусы Цельсия);

pressure – атмосферное давление (мм рт. ст.);

air humidity – влажность воздуха (проценты);

speed_wind – скорость ветра (м/с);

horizon_visibility – видимость (м);

hour – текущий час в сутках;

month – текущий месяц в году.

Обучение нейронной сети будет производиться по методике обучения с учителем, поэтому сформируем тестовые и тренировочные выборки. Для проверочного (тестового) набора оставить 20% данных.

Предсказывать будем значение мощности «P».

Для предикторов (наборов признаков, на основании которых, будет предсказываться значение «P») нужно провести нормализацию данных, так как признаки имеют различный масштаб. При этом, среднее значение каждого признака становится нулевым, а дисперсия равняется 1. Нормализация проводится с помощью формулы (4):

$$x_{i,j.\text{новое значение}} = \frac{x_{i,j.\text{старое значение}} - X_{j.\text{средн.}}}{\sigma_j}, \quad (4)$$

– где $x_{i,j.}$ – значение признака;

i – индекс строки;

j – номер столбца, т.е. рассматриваемый признак;

$X_{j.\text{средн.}}$ – среднее значение j-го признака;

σ_j – стандартное отклонение j-го признака.

На рисунок 24 представлен один из наборов данных после нормализации для тренировочной выборки

	T	pressure	air humidity	speed_wind	horizon_visibility	hour
0	-0.894502	1.537903	-0.303290	-1.093355	-0.949452	-1.541056
1	-1.179338	1.600200	-0.088483	-1.093355	-0.949452	-1.102540
2	-1.164347	1.600200	-0.356992	-1.093355	-0.949452	-0.664024
3	-0.991946	1.600200	-0.571800	0.349101	0.994548	-0.225508
4	-0.946972	1.512984	-0.195886	0.349101	0.994548	0.213008
...	...	...	...	...	...	...
1749	1.564079	0.242115	-1.431031	-1.093355	0.994548	1.090040
1750	1.151817	0.242115	-0.518098	-1.093355	-0.949452	1.528556
1751	0.971921	0.267034	-0.088483	-1.093355	-0.949452	-1.541056
1752	0.777033	0.316872	0.663344	-1.093355	0.778548	-1.102540
1753	0.986912	0.379169	0.502238	-0.372127	0.994548	-0.664024

Рисунок 24 – Нормализованные данные

3.3 Обучение нейронной сети

3.3.1 Рассмотрение различных моделей в зависимости от набора признаков

Из набора различных признаков (рисунок 25) было сформировано 33 различные модели, на которых оценивается полезность тех или иных признаков.

```
X0 = data[['T', 'pressure', 'hour']]
X1 = data[['T', 'pressure']]
X2 = data[['T', 'pressure', 'air humidity', 'hour']]
X3 = data[['T', 'pressure', 'speed_wind', 'hour']]
X4 = data[['T', 'pressure', 'horizon_visibility', 'hour']]
X5 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'hour']]
X6 = data[['T', 'pressure', 'air humidity', 'horizon_visibility', 'hour']]
X7 = data[['T', 'pressure', 'speed_wind', 'horizon_visibility', 'hour']]
X8 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour']]
X9 = data[['T', 'pressure', 'hour', 'month']]
X10 = data[['T', 'pressure', 'air humidity', 'hour', 'month']]
X11 = data[['T', 'pressure', 'speed_wind', 'hour', 'month']]
X12 = data[['T', 'pressure', 'horizon_visibility', 'hour', 'month']]
X13 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'hour', 'month']]
X14 = data[['T', 'pressure', 'air humidity', 'horizon_visibility', 'hour', 'month']]
X15 = data[['T', 'pressure', 'speed_wind', 'horizon_visibility', 'hour', 'month']]
X16 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour', 'month']]
X17 = data[['T', 'hour']]
X18 = data[['T', 'air humidity', 'hour']]
X19 = data[['T','speed_wind', 'hour']]
X20 = data[['T', 'horizon_visibility', 'hour']]
X21 = data[['T', 'air humidity', 'speed_wind', 'hour']]
X22 = data[['T', 'air humidity', 'horizon_visibility', 'hour']]
X23 = data[['T', 'speed_wind', 'horizon_visibility', 'hour']]
X24 = data[['T', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour']]
X25 = data[['T', 'hour', 'month']]
X26 = data[['T', 'air humidity', 'hour', 'month']]
X27 = data[['T', 'speed_wind', 'hour', 'month']]
X28 = data[['T', 'horizon_visibility', 'hour', 'month']]
X29 = data[['T', 'air humidity', 'speed_wind', 'hour', 'month']]
X30 = data[['T', 'air humidity', 'horizon_visibility', 'hour', 'month']]
X31 = data[['T', 'speed_wind', 'horizon_visibility', 'hour', 'month']]
X32 = data[['T', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour', 'month']]
```

Рисунок 25 – Наборы признаков для первичного отбора лучших моделей

Поставленная задача классифицируется, как задача регрессии, т.е. по входным данным нужно предсказать количественное значение искомой величины.

Обучение НС будет производить с помощью облачных вычислений Google Collaboratory. Ниже представлен код на языке Python:

```
#Создаём цикл для перебора всех вариаций признаков:
for j in range(0,len(X)):
  #Добавляем, в ранее созданный пустой список scores_all,
  #новые пустые списки для записи в них результатов j-то модели:
  scores_all.append([])
  #Создаём цикл для расчёта j-той модели 5 раз,
  #чтобы частично исключить результаты, вызванные переобучением:
  for i in range(0,5):

    #Создаём модель прямого распространения:
    model = Sequential()
    #Добавляем входной слой с функцией активации "relu",
    # "init='he_normal'" - данный способ инициализации весов, показывает лучшую сходимость
    #"2*input_n[j]" - количество нейронов на входном слое
    #"input_dim = input_n[j]" - размерность входных данных (кол-во признаков)
    model.add(Dense(2*input_n[j], input_dim = input_n[j], activation='relu', init='he_normal'))
    #Добавляем выходной слой с линейной функцие активации и 1 нейроном в слое
    model.add(Dense(1, activation='linear', init='he_normal'))

    #Настройка модели с помощью оптимизатора "RMSprop",
    #с функцией потерь "минимальная среднеквадратичная ошибка"
    #Результат оценивается по метрике "средняя абсолютная ошибка в процентах" для удобсва
 интерпритации
    model.compile(loss='mean_squared_error', optimizer='RMSprop' ,
                  metrics=['mean_absolute_percentage_error'])

    #Обучение модели, передаём тренировочные данные признаков и предикторов
    #Кол-во эпох равно 250, "batch_size=8" - размер пакетов для передачи нейронной сети,
    # чем меньше, медленнее обучается НС, но лучше результат
    #"verbose=0" - режим вывода результата эпох
    history = model.fit(X_train_values[j], y_train, epochs=250, batch_size=8, verbose=0)
    #Оценка результата модели на тестовой выборке:
    scores = model.evaluate(X_test_values[j], y_test)
    #добавляем в список "scores_all[j]" результаты i-того расчета модели
    scores_all[j].append(float("{:.2f}".format(scores[1])))
```

В результате обучения тридцати трех моделей получены оценки ошибок, представленные на рисунок 26. Столбцы – количество прогонов j модели. Строки – количество моделей (j=0,1…32).

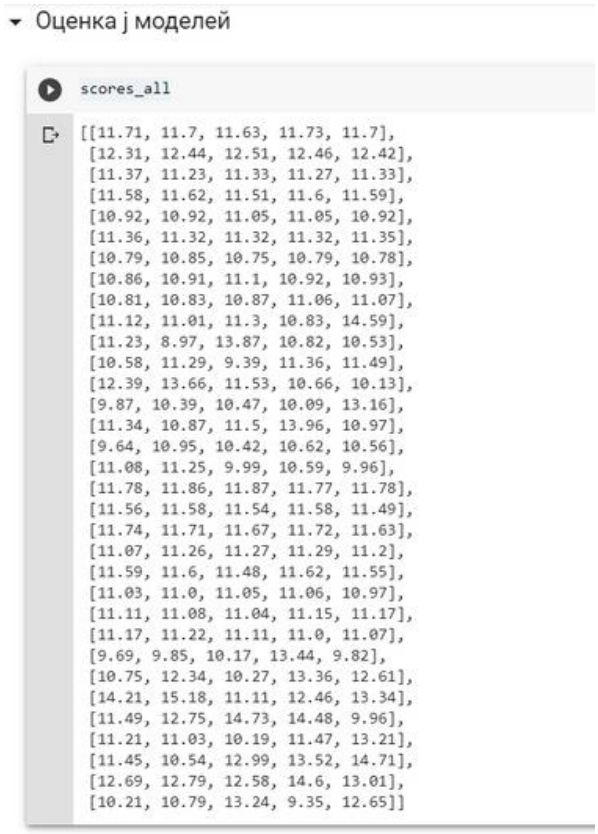


Рисунок 26 – Оценки ошибок первичных моделей

Приведем средние значения каждой модели на рисунок 27.

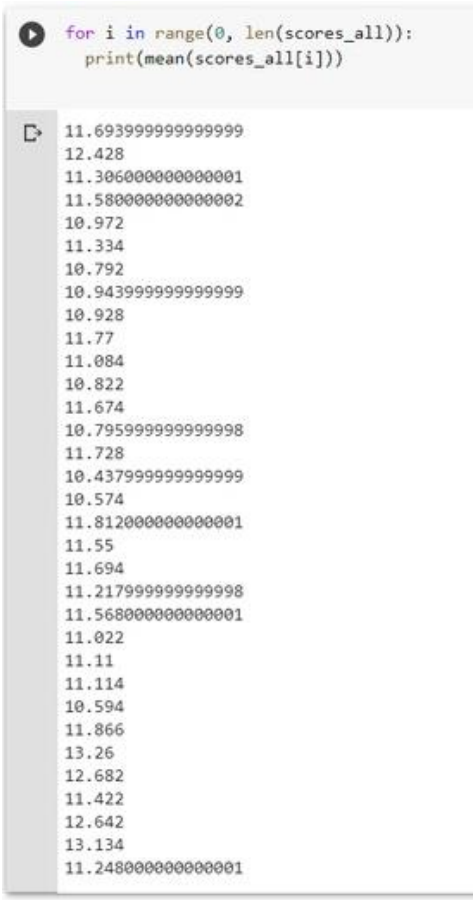


Рисунок 27 – Средние значения ошибок

Выберем 3 лучших модели по наименьшей ошибке (рисунок 26) и 5 лучших по среднему значению ошибки (рисунок 27), затем проанализируем эти модели.

По среднему значению ошибки:

1. Модель №6 (Ошибка 10,792%) – состоит из таких параметров, как температура, атмосферное давление, влажность воздуха, видимость, час;
2. Модель №10 (Ошибка 11,084%) – состоит из таких параметров, как температура, атмосферное давление, влажность воздуха, час, месяц;
3. Модель №11 (Ошибка 10,822%) – состоит из таких параметров, как температура, давление, скорость ветра, час, месяц;
4. Модель №13 (Ошибка 10,796%) – состоит из таких параметров, как температура, атмосферное давление, влажность воздуха, скорость ветра, час, месяц;
5. Модель №15 (Ошибка 10,438%) – состоит из таких параметров, как температура, атмосферное давление, скорость ветра, видимость, час, месяц.

По наименьшему значению ошибки:

6. Модель №16 (Ошибка 9,96%) – состоит из таких параметров, как температура, атмосферное давление, влажность воздуха, скорость ветра, видимость, час, месяц;
7. Модель №25 (Ошибка 9,69%) – состоит из таких параметров, как температура, час, месяц;
8. Модель №32 (Ошибка 9,35%) – состоит из таких параметров, как температура, влажность воздуха, скорость ветра, видимость, час, месяц.

Повторно проведем расчет этих 8-ми моделей, увеличив количество прогонов до 10 (рисунок 28, 29). Это позволит качественнее оценить выбранные модели и выбрать наилучшую.

```
X0 = data[['T', 'pressure', 'air humidity', 'horizon_visibility', 'hour']] #X6
X1 = data[['T', 'pressure', 'air humidity', 'hour', 'month']] #X10
X2 = data[['T', 'pressure', 'speed_wind', 'hour', 'month']] #X11
X3 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'hour', 'month']] #X13
X4 = data[['T', 'pressure', 'speed_wind', 'horizon_visibility', 'hour', 'month']] #X15
X5 = data[['T', 'pressure', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour', 'month']] #X16
X6 = data[['T', 'hour', 'month']] #X25
X7 = data[['T', 'air humidity', 'speed_wind', 'horizon_visibility', 'hour', 'month']] #X32
```

```
[ ]  scores_all
```

```
[[11.06, 10.73, 10.77, 10.94, 10.84, 10.67, 10.88, 10.97, 10.84, 10.81],
 [9.5, 9.65, 14.78, 13.13, 10.91, 10.24, 9.74, 14.42, 10.34, 15.93],
 [10.33, 10.35, 10.28, 10.65, 9.49, 11.02, 11.11, 9.33, 9.91, 12.56],
 [11.81, 10.23, 11.72, 10.61, 10.92, 11.36, 10.72, 11.05, 11.33, 10.4],
 [11.14, 12.06, 11.45, 12.29, 14.71, 10.29, 10.42, 12.76, 11.46, 11.84],
 [11.33, 11.97, 12.4, 10.74, 11.12, 12.45, 9.98, 11.41, 13.16, 10.11],
 [12.42, 15.31, 12.5, 11.72, 13.17, 11.9, 10.08, 12.53, 10.15, 10.66],
 [11.05, 13.07, 10.97, 11.13, 12.85, 11.28, 10.21, 12.51, 12.7, 11.33],
```

Рисунок 28 – Оценки ошибок вторичных моделей

Вычислим средние значения ошибок каждой модели (рисунок 29)

```
10.851
11.863
10.503
11.015
11.842
11.467
12.044
11.71
```

Рисунок 29 – Средние значения ошибок вторичных моделей

Анализируя значения ошибок (см. рисунок 28) и средние значения ошибок вторичных моделей (см. рисунок 29), выберем модель №11 со средним значением ошибки, равным 10,503% и самым минимальны значение ошибки в одном из прогонов, равным 9,33%.

3.3.2 Поиск наилучшей модели, учитывающей погодные факторы

Следующей задачей является уменьшение значения ошибки выбранной модели. Это достигается следующим образом:

- Изменение количества входных нейронов;
- Изменение функции активации;
- Изменение оптимизаторов;
- Добавление внутренних слоёв.

Для автоматического перебора исследуемых гиперпараметров ниже представлен код (рисунок 30), который написан с помощью руководства библиотеки Keras [6].

```python
def NN_depth(depth):
  if depth == 1:
    return model.add(Dense(multiplier[k] * input_n[j], input_dim = input_n[j], activation=activations[l], init='he_normal'))
  elif depth == 2:
    first_layer = model.add(Dense(multiplier[k] * input_n[j], input_dim = input_n[j], activation=activations[l], init='he_normal')),
    second_layer = model.add(Dense(int(multiplier_2[k] * input_n[j]), activation=activations[l], init='he_normal'))
    return first_layer, second_layer

  elif depth == 3:
    first_layer = model.add(Dense(multiplier[k] * input_n[j], input_dim = input_n[j], activation=activations[l], init='he_normal')),
    second_layer = model.add(Dense(int(multiplier_2[k] * input_n[j]), activation=activations[l], init='he_normal'))
    third_layer = model.add(Dense(int(multiplier_3[k] * input_n[j]), activation=activations[l], init='he_normal'))
    return first_layer, second_layer, third_layer
```

```python
multiplier = [1, 2, 10, 100, 150, 200, 1000] #  множитель кол-ва нейронов 1 слоя
multiplier_2 = [0.5, 1, 5, 50, 70, 100, 500] # множитель кол-ва нейронов 2 слоя
multiplier_3 = [0.25, 0.5, 2.5, 25, 35, 50, 250] # множитель кол-ва нейронов 3 слоя
depths = [1, 2, 3] # Количество слоёв нейронной сети
activations = ['relu', 'elu'] # набор функций активации
optimizers = ['RMSprop', 'adam'] # набор оптимизаторов
for p in range (0, len(depths)): # цикл отвечает за глубину НС
  for m in range (0, len(optimizers)): # цикл отвечает за перебор оптимизаторов
    for l in range(0, len(activations)): # цикл отвечает за перебор функций активации
      # Training a model
      for k in range(0, len(multiplier)): # цикл отвечает за перебор кол-ва нейронов во входном слое
        scores_all=[]
        for j in range(0,1): # цикл отвечате за перебор входных данных
          scores_all.append([])
          for i in range(0,3): # цикл отвечает за повторный прогон одной и той же модели
            # Создание модели:
            model = Sequential()
            NN_depth(depths[p])
            model.add(Dense(1, activation='linear', init='he_normal'))
            # Компиляция модели:
            model.compile(loss='mean_squared_error', optimizer=optimizers[m] , metrics=['mean_absolute_percentage_error'])
            history = model.fit(X_train_values[j], y_train, epochs=200, batch_size=8, verbose=0)
            # Формирование списка ошибок:
            scores = model.evaluate(X_test_values[j], y_test)
            scores_all[j].append(float("{:.2f}".format(scores[1])))
            # Вывод результата:
          print("Расчет модели с кол-вом нейронов во входном слое = ",
                multiplier[k]*input_n[j], ',', ' функцией активации - ',
                activations[l], ",", ' оптимизатором - ', optimizers[m],
                ",", " с кол-вом слоёв = ", depths[p],
                " закончен с результатами:", sep='')
        print(scores_all)
```

Рисунок 30 – Код для автоматического перебора гиперпараметров

3.3.2.1 Однослойная нейронная сеть

1. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для модели с функцией активации – «ReLU» и оптимизатором «RMSprop» (рисунок 31):

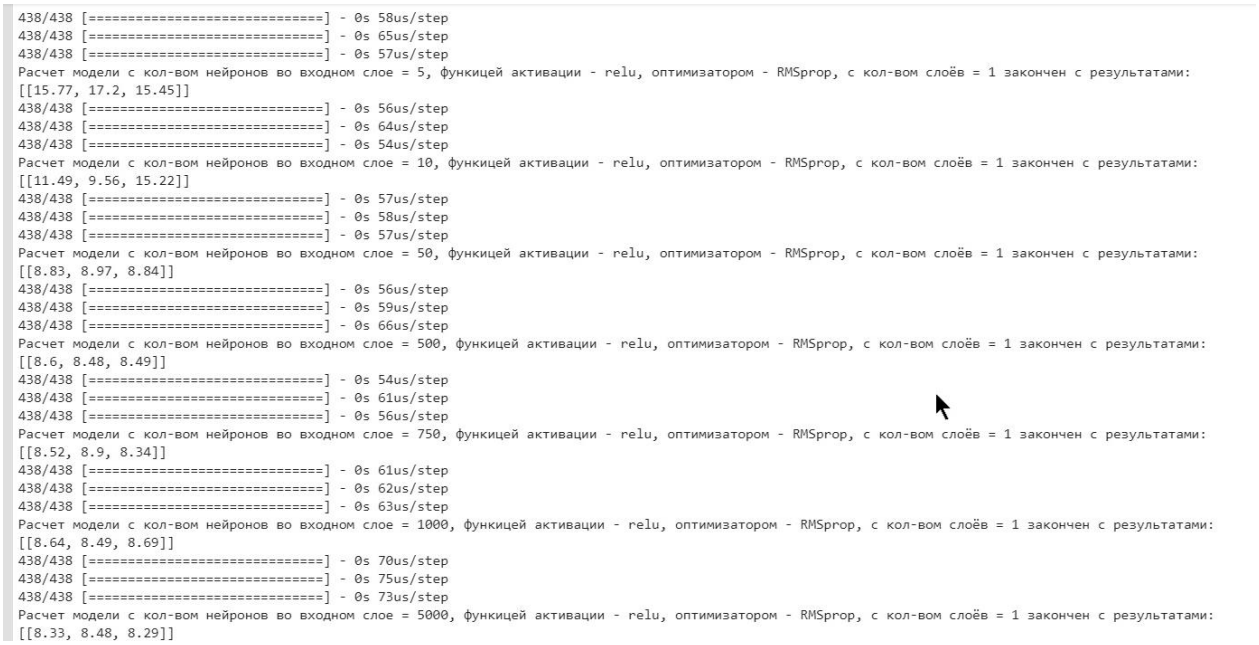


Рисунок 31 – ReLU, RMSprop, однослойная НС

2. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для модели с функцией активации – «ELU» и оптимизатором «RMSprop» (рисунок 32):

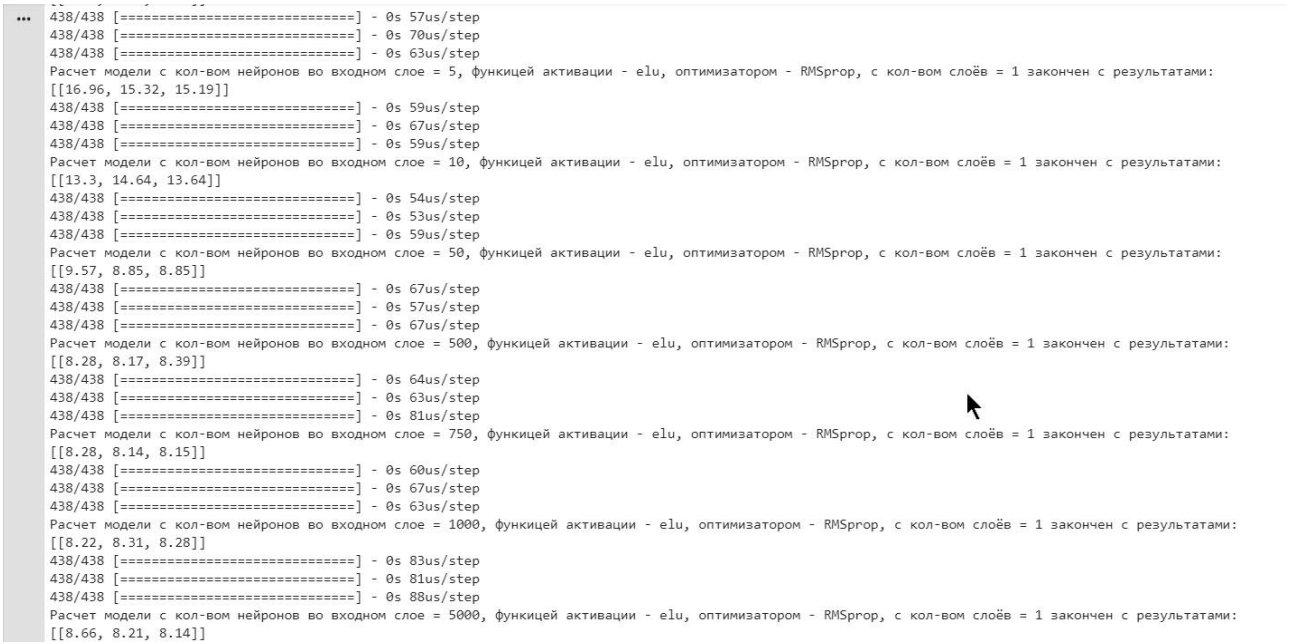


Рисунок 32 – ELU, RMSprop, однослойная НС

3. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для модели с функцией активации – «ReLU» и оптимизатором «Adam» (рисунок 33):

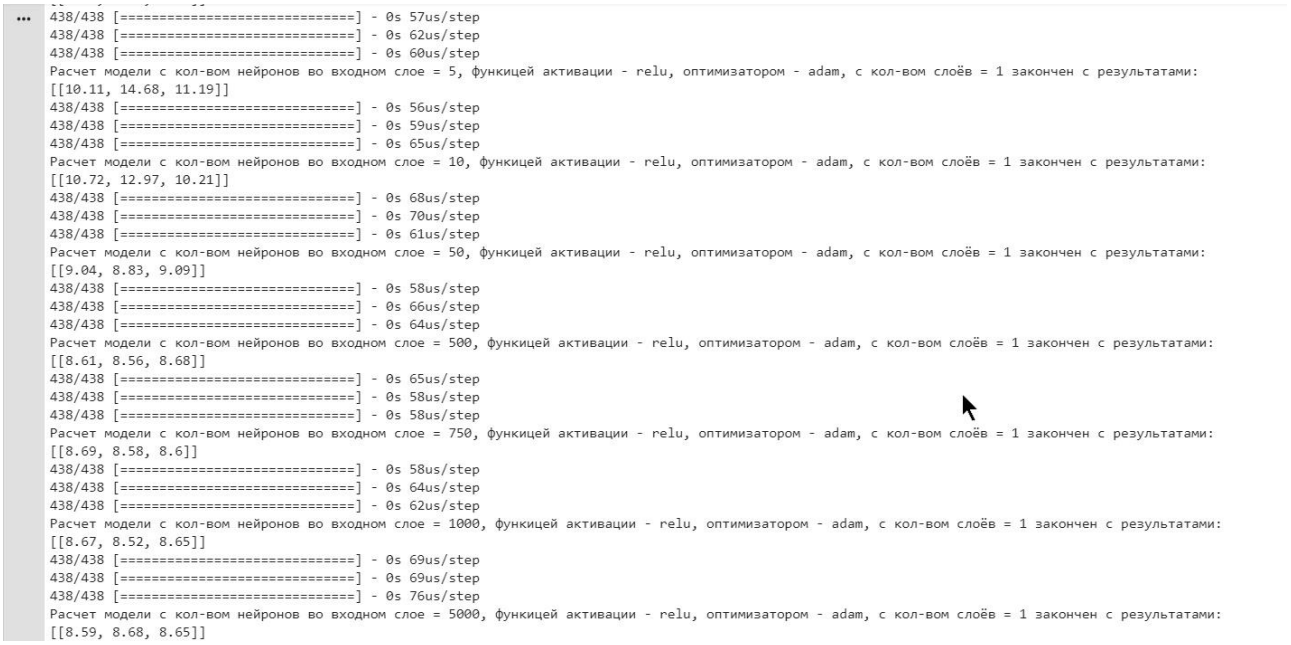


Рисунок 33 – ReLU, Adam, однослойная НС

4. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для модели с функцией активации – «ELU» и оптимизатором «Adam» (рисунок 34):

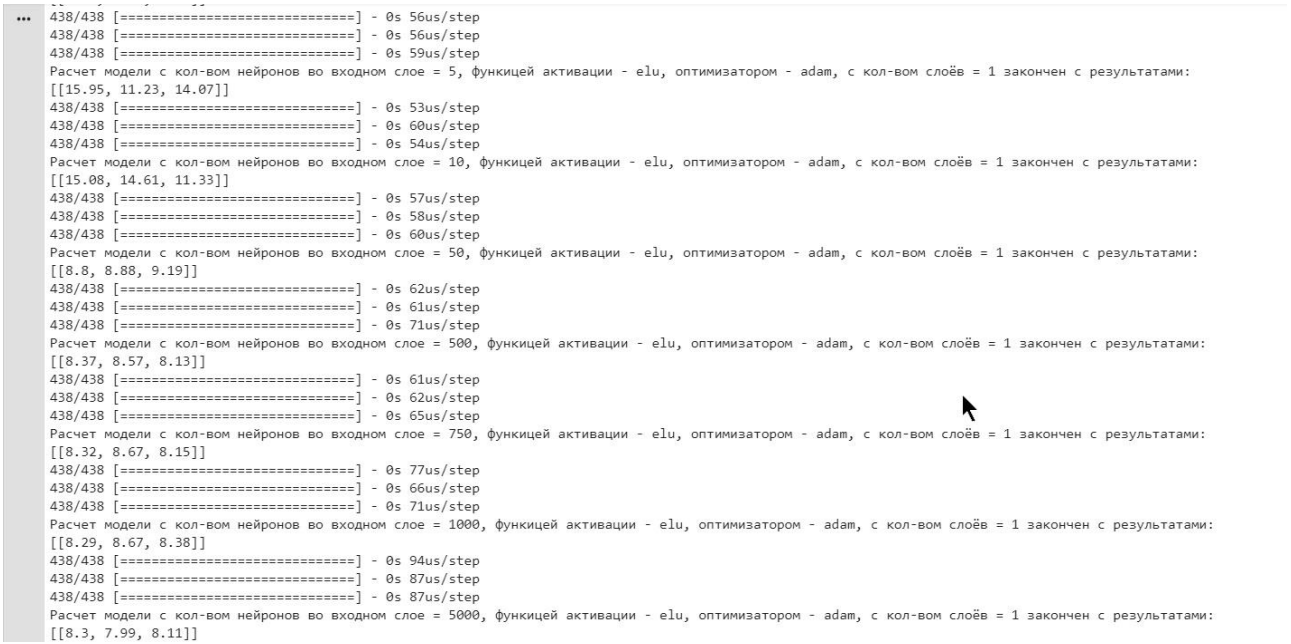


Рисунок 34 – ELU, Adam, однослойная НС

3.3.2.2 Двухслойная НС

1. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для двухслойной модели с функцией активации – «ReLU» и оптимизатором «RMSprop» (рисунок 35):

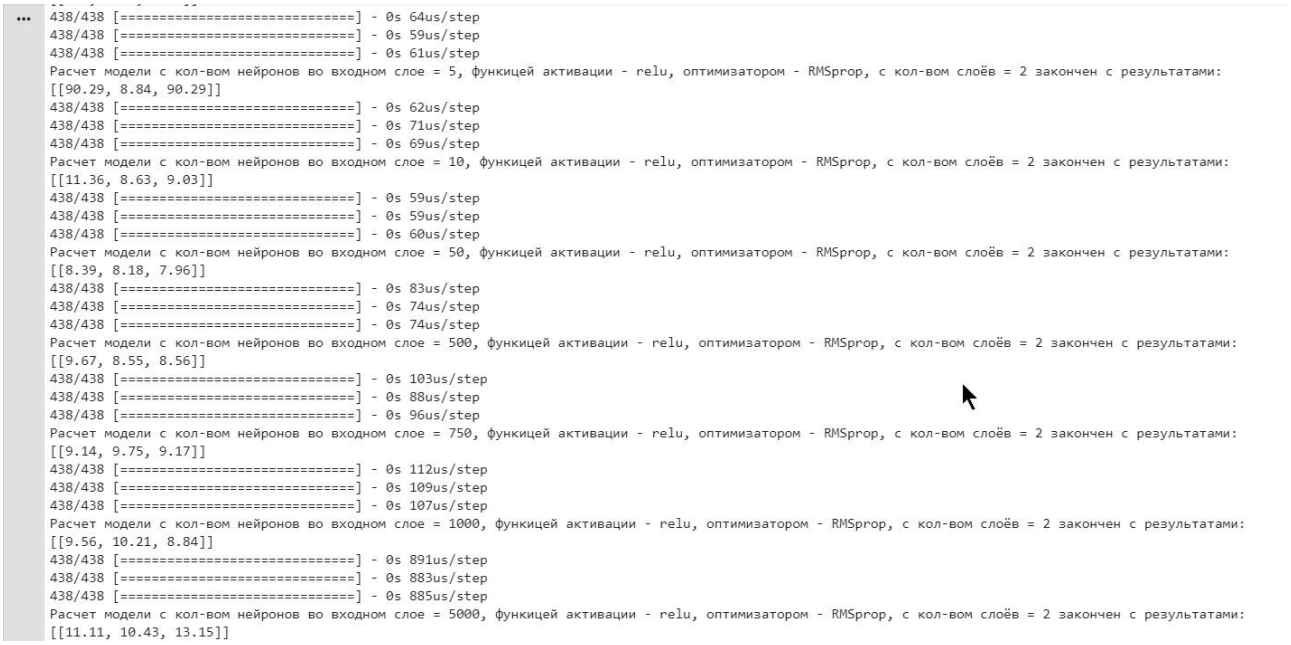


Рисунок 35 – ReLU, RMSprop, двухслойная НС

2. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для двухслойной модели с функцией активации – «ELU» и оптимизатором «RMSprop» (рисунок 36):

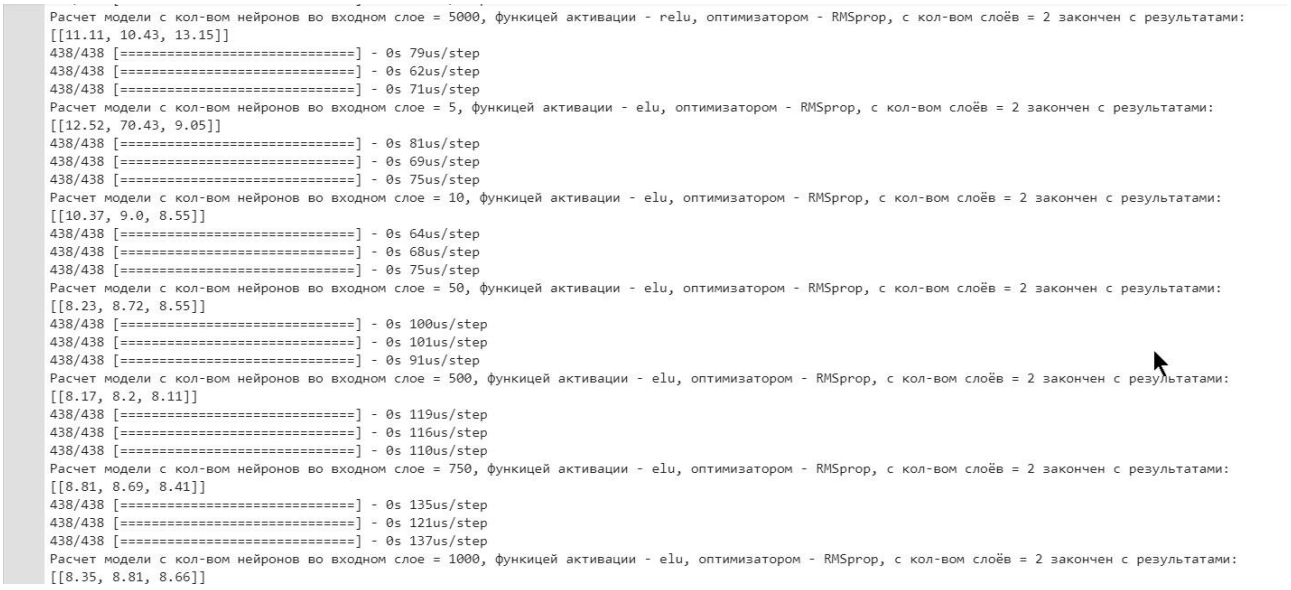


Рисунок 36 – ELU, RMSprop, двухслойная НС

3. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для двухслойной модели с функцией активации – «ReLU» и оптимизатором «Adam» (рисунок 37):

```
438/438 [==============================] - 0s 59us/step
438/438 [==============================] - 0s 56us/step
438/438 [==============================] - 0s 61us/step
Расчет модели с кол-вом нейронов во входном слое = 5, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[11.21, 8.85, 90.31]]
438/438 [==============================] - 0s 50us/step
438/438 [==============================] - 0s 59us/step
438/438 [==============================] - 0s 54us/step
Расчет модели с кол-вом нейронов во входном слое = 10, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.74, 10.01, 9.21]]
438/438 [==============================] - 0s 72us/step
438/438 [==============================] - 0s 59us/step
438/438 [==============================] - 0s 58us/step
Расчет модели с кол-вом нейронов во входном слое = 50, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.46, 7.97, 9.14]]
438/438 [==============================] - 0s 69us/step
438/438 [==============================] - 0s 74us/step
438/438 [==============================] - 0s 68us/step
Расчет модели с кол-вом нейронов во входном слое = 500, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.58, 9.21, 8.87]]
438/438 [==============================] - 0s 76us/step
438/438 [==============================] - 0s 76us/step
438/438 [==============================] - 0s 83us/step
Расчет модели с кол-вом нейронов во входном слое = 750, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.8, 8.6, 9.85]]
438/438 [==============================] - 0s 100us/step
438/438 [==============================] - 0s 92us/step
438/438 [==============================] - 0s 86us/step
Расчет модели с кол-вом нейронов во входном слое = 1000, функцией активации - relu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[9.21, 9.03, 8.65]]
```

Рисунок 37 – ReLU, Adam, двухслойная НС

4. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для двухслойной модели с функцией активации – «ELU» и оптимизатором «Adam» (рисунок 38):

```
438/438 [==============================] - 0s 55us/step
438/438 [==============================] - 0s 54us/step
438/438 [==============================] - 0s 58us/step
Расчет модели с кол-вом нейронов во входном слое = 5, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[10.79, 15.19, 70.41]]
438/438 [==============================] - 0s 57us/step
438/438 [==============================] - 0s 61us/step
438/438 [==============================] - 0s 65us/step
Расчет модели с кол-вом нейронов во входном слое = 10, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.7, 9.25, 8.91]]
438/438 [==============================] - 0s 57us/step
438/438 [==============================] - 0s 56us/step
438/438 [==============================] - 0s 56us/step
Расчет модели с кол-вом нейронов во входном слое = 50, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.31, 8.14, 8.31]]
438/438 [==============================] - 0s 70us/step
438/438 [==============================] - 0s 68us/step
438/438 [==============================] - 0s 70us/step
Расчет модели с кол-вом нейронов во входном слое = 500, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.22, 8.1, 8.59]]
438/438 [==============================] - 0s 81us/step
438/438 [==============================] - 0s 83us/step
438/438 [==============================] - 0s 79us/step
Расчет модели с кол-вом нейронов во входном слое = 750, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.26, 8.05, 8.07]]
438/438 [==============================] - 0s 98us/step
438/438 [==============================] - 0s 99us/step
438/438 [==============================] - 0s 102us/step
Расчет модели с кол-вом нейронов во входном слое = 1000, функцией активации - elu, оптимизатором - adam, с кол-вом слоёв = 2 закончен с результатами:
[[8.69, 8.38, 9.62]]
```

Рисунок 38 – ELU, Adam, двухслойная НС

3.3.2.3 Трёхслойная НС

1. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для трёхслойной модели с функцией активации – «ReLU» и оптимизатором «Adam» (рисунок 39):

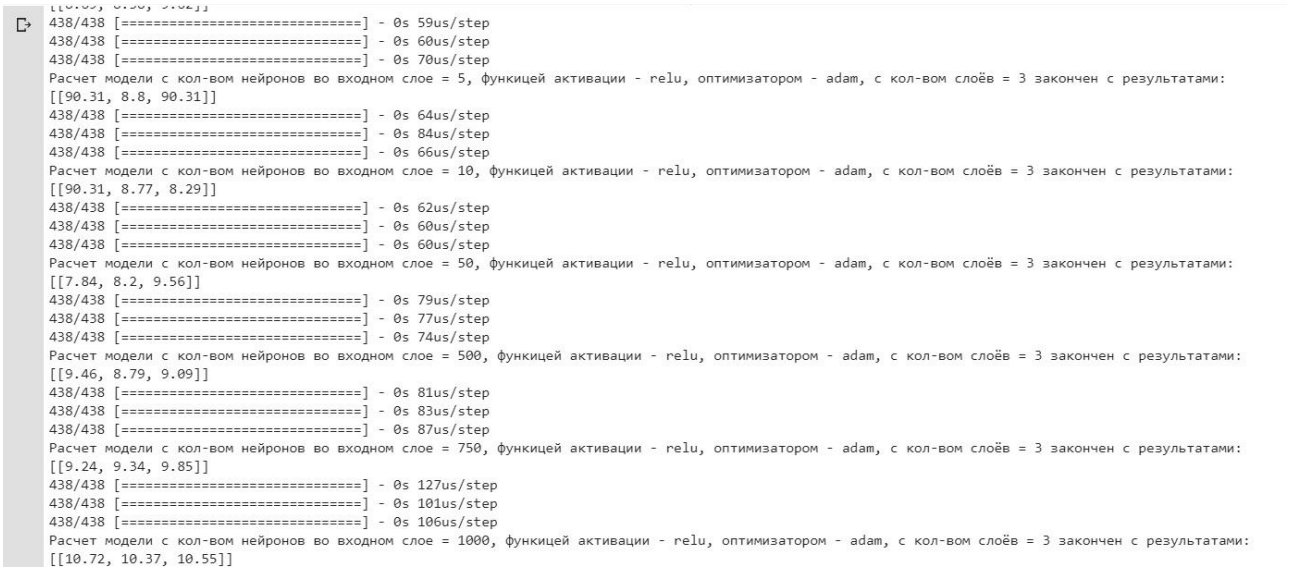


Рисунок 39 – ReLU, Adam, трёхслойная НС

2. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для трёхслойной модели с функцией активации – «ELU» и оптимизатором «Adam» (рисунок 40):

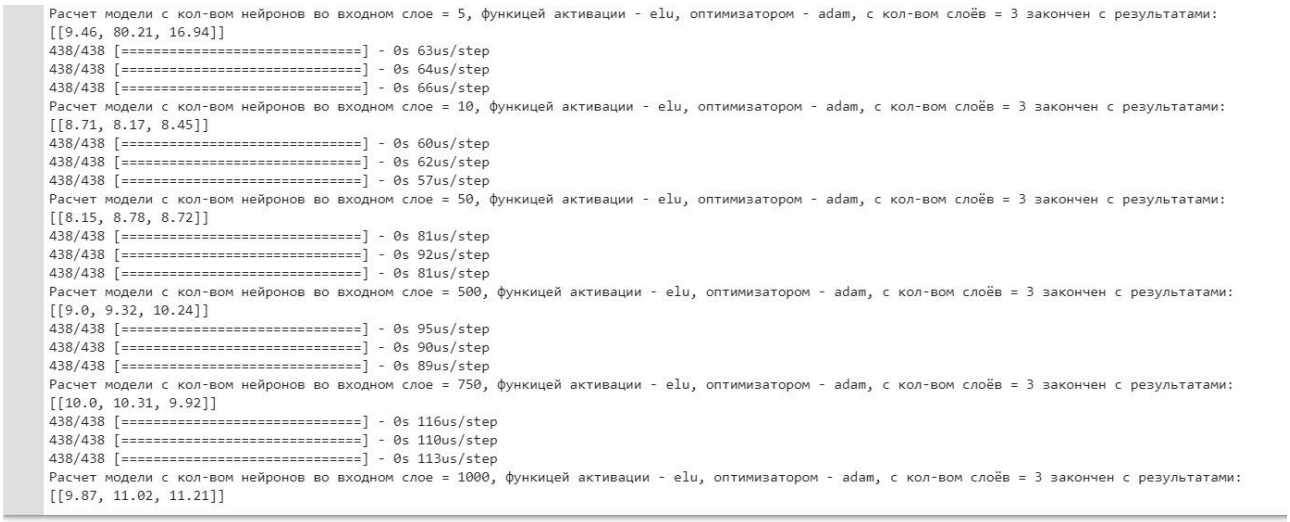


Рисунок 40 – ELU, Adam, трёхслойная НС

3. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для трёхслойной модели с функцией активации – «ReLU» и оптимизатором «RMSprop» (рисунок 41):

```
438/438 [==============================] - 0s 79us/step
438/438 [==============================] - 0s 81us/step
438/438 [==============================] - 0s 77us/step
Расчет модели с кол-вом нейронов во входном слое = 5, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[90.29, 90.29, 90.29]]
438/438 [==============================] - 0s 67us/step
438/438 [==============================] - 0s 71us/step
438/438 [==============================] - 0s 73us/step
Расчет модели с кол-вом нейронов во входном слое = 10, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[90.29, 8.49, 9.32]]
438/438 [==============================] - 0s 84us/step
438/438 [==============================] - 0s 71us/step
438/438 [==============================] - 0s 67us/step
Расчет модели с кол-вом нейронов во входном слое = 50, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[9.64, 8.25, 8.43]]
438/438 [==============================] - 0s 75us/step
438/438 [==============================] - 0s 82us/step
438/438 [==============================] - 0s 86us/step
Расчет модели с кол-вом нейронов во входном слое = 500, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[11.04, 9.96, 9.22]]
438/438 [==============================] - 0s 90us/step
438/438 [==============================] - 0s 88us/step
438/438 [==============================] - 0s 93us/step
Расчет модели с кол-вом нейронов во входном слое = 750, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[10.85, 10.75, 10.29]]
438/438 [==============================] - 0s 116us/step
438/438 [==============================] - 0s 115us/step
438/438 [==============================] - 0s 106us/step
Расчет модели с кол-вом нейронов во входном слое = 1000, функцией активации - relu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[8.88, 10.1, 10.88]]
```

Рисунок 41 – ReLU, RMSprop, трёхслойная НС

4. Рассмотрим изменение ошибки в зависимости от количества нейронов во входном слое для трёхслойной модели с функцией активации – «ELU» и оптимизатором «RMSprop» (рисунок 42):

```
Расчет модели с кол-вом нейронов во входном слое = 5, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[14.05, 14.19, 80.38]]
438/438 [==============================] - 0s 58us/step
438/438 [==============================] - 0s 63us/step
438/438 [==============================] - 0s 62us/step
Расчет модели с кол-вом нейронов во входном слое = 10, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[8.08, 8.47, 8.22]]
438/438 [==============================] - 0s 63us/step
438/438 [==============================] - 0s 63us/step
438/438 [==============================] - 0s 66us/step
Расчет модели с кол-вом нейронов во входном слое = 50, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[8.89, 8.67, 9.56]]
438/438 [==============================] - 0s 80us/step
438/438 [==============================] - 0s 85us/step
438/438 [==============================] - 0s 84us/step
Расчет модели с кол-вом нейронов во входном слое = 500, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[10.63, 11.38, 10.22]]
438/438 [==============================] - 0s 96us/step
438/438 [==============================] - 0s 95us/step
438/438 [==============================] - 0s 86us/step
Расчет модели с кол-вом нейронов во входном слое = 750, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[12.19, 13.26, 13.46]]
438/438 [==============================] - 0s 123us/step
438/438 [==============================] - 0s 114us/step
438/438 [==============================] - 0s 119us/step
Расчет модели с кол-вом нейронов во входном слое = 1000, функцией активации - elu, оптимизатором - RMSprop, с кол-вом слоёв = 3 закончен с результатами:
[[11.46, 14.09, 21.83]]
```

Рисунок 42 – ELU, RMSprop, трёхслойная НС

Проанализировав ошибки различных моделей (рисунки 31-42), можно сделать вывод: добавление скрытых слоёв практические не улучшает показатель точности модели, а иногда наоборот – увеличивает ошибку.

Для примера построения результатов прогнозирования выберем лучшую модель из всех тестов, с MAPE равным 7,99% (см. рисунки 31 - 42), со следующими гиперпараметрами:

- Однослойная нейронная сеть с 5000 нейронами во входном слое;
- Функция активации – ELU;

- Оптимизатор – Adam.

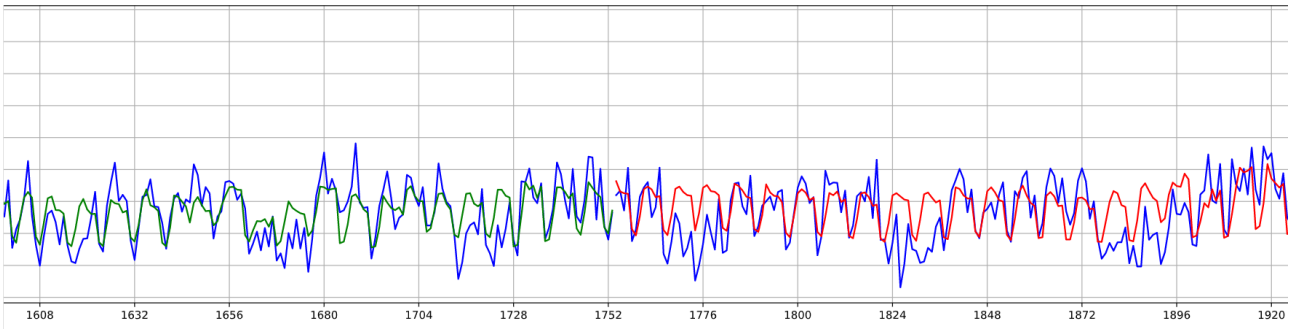


Рисунок 43 – Результат прогнозирования

Красным цветом построен график предсказанных значений, результат оставляет желать лучшего, поэтому для более точного прогнозирования рассмотрим следующие способы.

3.3.3 Рекуррентная нейронная сеть LSTM

Для обучения рекуррентной НС потребуются только данные нагрузок (см. рисунок 22). После преобразования их в формат DataFrame они будут иметь следующий вид на рисунок 44:

	time	P
0	2008-01-01 00:00:00	513.705872
1	2008-01-01 01:00:00	475.052185
2	2008-01-01 02:00:00	509.094910
3	2008-01-01 03:00:00	453.152588
4	2008-01-01 04:00:00	480.089600
5	2008-01-01 05:00:00	469.061890
6	2008-01-01 06:00:00	479.227905
7	2008-01-01 07:00:00	473.359772
8	2008-01-01 08:00:00	456.218994
9	2008-01-01 09:00:00	466.914185
10	2008-01-01 10:00:00	468.043793

Рисунок 44 – Данные нагрузок в формате DataFrame

На рисунке 45 представлен код модели, который написан с помощью руководства библиотеки TensorFlow [7].

```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(24, return_sequences=True, input_shape=x_train_uni.shape[-2:]),
    tf.keras.layers.LSTM(24, activation='relu'),
    tf.keras.layers.Dense(24)
])


#model.compile(optimizer=tf.keras.optimizers.RMSprop(clipvalue=1.0), loss='mean_squared_error', metrics=['mean_absolute_percentage_error'])
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_percentage_error'])


EVALUATION_INTERVAL = 400
EPOCHS = int(lenth / BATCH_SIZE)

model.fit(train_univariate, epochs=EPOCHS,
                      steps_per_epoch=EVALUATION_INTERVAL,
                      validation_data=val_univariate, validation_steps=50,
                      callbacks=[reduce_lr, m_checkpoint])
```

Рисунок 45 – Код для обучения LSTM сети

```
Epoch 178/200
400/400 [==============================] - 14s 36ms/step - loss: 1704.1184 - mean_absolute_percentage_error: 6.4124 - val_loss: 2120.8616 - val_mean_absolute_percentage_error: 7.8762 - lr: 1.0000e-05
Epoch 179/200
400/400 [==============================] - 14s 35ms/step - loss: 1721.5970 - mean_absolute_percentage_error: 6.4054 - val_loss: 2113.2412 - val_mean_absolute_percentage_error: 8.0203 - lr: 1.0000e-05
Epoch 180/200
400/400 [==============================] - 14s 36ms/step - loss: 1698.7307 - mean_absolute_percentage_error: 6.3694 - val_loss: 2121.2886 - val_mean_absolute_percentage_error: 8.1356 - lr: 1.0000e-05
Epoch 181/200
400/400 [==============================] - 14s 35ms/step - loss: 1726.3943 - mean_absolute_percentage_error: 6.4180 - val_loss: 2102.6484 - val_mean_absolute_percentage_error: 8.0034 - lr: 1.0000e-05
Epoch 182/200
400/400 [==============================] - 14s 35ms/step - loss: 1710.0598 - mean_absolute_percentage_error: 6.4008 - val_loss: 2149.3276 - val_mean_absolute_percentage_error: 8.2168 - lr: 1.0000e-05
Epoch 183/200
400/400 [==============================] - 14s 36ms/step - loss: 1714.7800 - mean_absolute_percentage_error: 6.4184 - val_loss: 2094.5093 - val_mean_absolute_percentage_error: 7.9427 - lr: 1.0000e-05
Epoch 184/200
400/400 [==============================] - 14s 35ms/step - loss: 1740.2338 - mean_absolute_percentage_error: 6.4395 - val_loss: 2122.3843 - val_mean_absolute_percentage_error: 8.1406 - lr: 1.0000e-05
Epoch 185/200
400/400 [==============================] - 15s 36ms/step - loss: 1667.0519 - mean_absolute_percentage_error: 6.3168 - val_loss: 2097.5881 - val_mean_absolute_percentage_error: 8.0588 - lr: 1.0000e-05
Epoch 186/200
400/400 [==============================] - 14s 36ms/step - loss: 1713.5051 - mean_absolute_percentage_error: 6.4254 - val_loss: 2115.1440 - val_mean_absolute_percentage_error: 8.0272 - lr: 1.0000e-05
Epoch 187/200
400/400 [==============================] - 15s 37ms/step - loss: 1698.6884 - mean_absolute_percentage_error: 6.3599 - val_loss: 2112.6355 - val_mean_absolute_percentage_error: 8.0564 - lr: 1.0000e-05
Epoch 188/200
400/400 [==============================] - 14s 35ms/step - loss: 1701.4546 - mean_absolute_percentage_error: 6.3681 - val_loss: 2086.4866 - val_mean_absolute_percentage_error: 7.9907 - lr: 1.0000e-05
Epoch 189/200
400/400 [==============================] - 14s 35ms/step - loss: 1682.1406 - mean_absolute_percentage_error: 6.3635 - val_loss: 2087.4062 - val_mean_absolute_percentage_error: 7.9367 - lr: 1.0000e-05
Epoch 190/200
400/400 [==============================] - 14s 36ms/step - loss: 1732.6287 - mean_absolute_percentage_error: 6.4407 - val_loss: 2129.4937 - val_mean_absolute_percentage_error: 8.2271 - lr: 1.0000e-05
Epoch 191/200
400/400 [==============================] - 14s 36ms/step - loss: 1690.4735 - mean_absolute_percentage_error: 6.3486 - val_loss: 2089.1572 - val_mean_absolute_percentage_error: 7.8283 - lr: 1.0000e-05
Epoch 192/200
400/400 [==============================] - 14s 35ms/step - loss: 1695.6002 - mean_absolute_percentage_error: 6.3607 - val_loss: 2066.8210 - val_mean_absolute_percentage_error: 7.8858 - lr: 1.0000e-05
Epoch 193/200
400/400 [==============================] - 14s 35ms/step - loss: 1687.8497 - mean_absolute_percentage_error: 6.3596 - val_loss: 2085.7034 - val_mean_absolute_percentage_error: 7.8808 - lr: 1.0000e-05
Epoch 194/200
400/400 [==============================] - 14s 35ms/step - loss: 1713.5591 - mean_absolute_percentage_error: 6.3899 - val_loss: 2174.3542 - val_mean_absolute_percentage_error: 8.2769 - lr: 1.0000e-05
Epoch 195/200
400/400 [==============================] - 14s 35ms/step - loss: 1688.7469 - mean_absolute_percentage_error: 6.3411 - val_loss: 2079.2898 - val_mean_absolute_percentage_error: 7.9757 - lr: 1.0000e-05
Epoch 196/200
400/400 [==============================] - 14s 35ms/step - loss: 1694.8058 - mean_absolute_percentage_error: 6.3636 - val_loss: 2112.5137 - val_mean_absolute_percentage_error: 8.1117 - lr: 1.0000e-05
Epoch 197/200
400/400 [==============================] - 14s 35ms/step - loss: 1718.6055 - mean_absolute_percentage_error: 6.4172 - val_loss: 2072.3843 - val_mean_absolute_percentage_error: 7.8857 - lr: 1.0000e-05
Epoch 198/200
400/400 [==============================] - 14s 36ms/step - loss: 1691.2336 - mean_absolute_percentage_error: 6.3507 - val_loss: 2127.9023 - val_mean_absolute_percentage_error: 8.1621 - lr: 1.0000e-05
Epoch 199/200
400/400 [==============================] - 14s 35ms/step - loss: 1707.3517 - mean_absolute_percentage_error: 6.3811 - val_loss: 2079.2808 - val_mean_absolute_percentage_error: 7.9308 - lr: 1.0000e-05
Epoch 200/200
400/400 [==============================] - 14s 35ms/step - loss: 1699.1116 - mean_absolute_percentage_error: 6.3862 - val_loss: 2071.5034 - val_mean_absolute_percentage_error: 7.9301 - lr: 1.0000e-05
'\nmodel.fit(train_univariate, epochs=EPOCHS,\n                      steps_per_epoch=EVALUATION_INTERVAL,\n                      validation_data=val_univariate, validation_steps=50)\n'
```

Рисунок 46 – Результат обучения LSTM сети

Значение val_mean_absolute_percentage_error обозначает MAPE модели на тестовой выборке на данной эпохе. Полную конфигурацию модели на данной эпохе можно сохранить с помощью функций обратного вызова (callbacks) в библиотеке Keras. Поэтому можно, считать, что данный способ позволяет достичь ошибку около 7, 8%, что соизмеримо с предыдущим способом. Однако, данный результат у LSTM сети достигается спустя 2865 секунд, против 60 секунд у сети прямого распространения.

3.3.4 Обзор наилучшего результата среди всех проведенных опытов

3.3.4.1 Подготовка данных

В качестве исходных данных используются только данные о почасовом изменении мощности в рассматриваем узле энергосистемы (см. рисунок 22, 44).

В данном случает, задача сводится к прогнозированию временных рядов с помощью одномерных данных, но все данные разбиваются на интервалы периодичности. Так как графики нагрузок имеют периодичность 24 часа, то мы всю выборку разбиваем на части по 24 часа, а нейронная сеть уже будет обучаться на предсказывании 25-го часа (рисунок 47). Сезонность учитывать не нужно так как, в качестве тренировочной выборки загружается первые 80% данных, а в качестве тестовой следующие 20% (с ними будет сравниваться прогноз нейронной сети). Если же разбить выборки случайным образом, то пришлось бы учитывать сезонность. Так как данных очень много, данный способ, косвенно будет учитывать изменения погодных факторов, социальных, экономических, которые уже будут отражены в изменении мощности предыдущих показаний.

	t-24	t-23	t-22	t-21	t-20	t-19	t-18	t-17	t-16	t-15	t-14	t-13	t-12
24	513.705872	475.052185	509.094910	453.152588	480.089600	469.061890	479.227905	473.359772	456.218994	466.914185	468.043793	466.976196	470.323364
25	475.052185	509.094910	453.152588	480.089600	469.061890	479.227905	473.359772	456.218994	466.914185	468.043793	466.976196	470.323364	459.326508
26	509.094910	453.152588	480.089600	469.061890	479.227905	473.359772	456.218994	466.914185	468.043793	466.976196	470.323364	459.326508	452.678894
27	453.152588	480.089600	469.061890	479.227905	473.359772	456.218994	466.914185	468.043793	466.976196	470.323364	459.326508	452.678894	456.765076
28	480.089600	469.061890	479.227905	473.359772	456.218994	466.914185	468.043793	466.976196	470.323364	459.326508	452.678894	456.765076	463.698669

t-11	t-10	t-9	t-8	t-7	t-6	t-5	t-4	t-3	t-2	t-1	t
459.326508	452.678894	456.765076	463.698669	507.184998	526.304932	560.559570	536.710815	548.342529	565.735962	531.260620	519.970703
452.678894	456.765076	463.698669	507.184998	526.304932	560.559570	536.710815	548.342529	565.735962	531.260620	519.970703	503.434082
456.765076	463.698669	507.184998	526.304932	560.559570	536.710815	548.342529	565.735962	531.260620	519.970703	503.434082	469.688293
463.698669	507.184998	526.304932	560.559570	536.710815	548.342529	565.735962	531.260620	519.970703	503.434082	469.688293	489.081787
507.184998	526.304932	560.559570	536.710815	548.342529	565.735962	531.260620	519.970703	503.434082	469.688293	489.081787	453.513702

Рисунок 47 – Данные, подаваемые на входной слой НС

3.3.4.2 Код и архитектура

Выбрана архитектура нейронной сети – прямого распространения (см. рисунок 16), нейронная сеть состоит из трёх слоёв:

- Входного, с 48-ю нейронами в слое (функция активации – ReLU);
- Скрытого, с 48-ю нейронами в слое (функция активации – ReLU);
- Выходного (состоит из одного нейрона, на который с предыдущего слоя подаются значения, функция активации – линейная).

Оптимизатор – RMSprop. Для данной модели MAPE (средняя абсолютная ошибка в процентах) получилась 4,59 %. По результатам прогнозирования построен график (см. Приложение А). Точный прогноз получился на 1356 часов (примерно 2 месяца).

```
multiplier = [1, 2, 10, 100, 200, 1000]
activations = ['relu', 'elu']
optimizers = ['adam', 'RMSprop']

for m in range (0, len(optimizers)): # цикл отвечает за перебор оптимизаторов
  for l in range(0, len(activations)): #цикл отвечает за перебор функций активации
    # Training a model
    for k in range(0, len(multiplier)): #цикл отвечает за перебор кол-ва нейронов во входном слое
      scores_all=[]
      for j in range(0,1): # цикл отвечате за перебор входных данных
        scores_all.append([])
        for i in range(0,3): # цикл отвечает за повторный прогон одной и той же модели
          # Creating a model
          model = Sequential()
          model.add(Dense(multiplier[k] * 24, input_dim = 24, activation=activations[l], init='he_normal'))
          model.add(Dense(multiplier[k] * 24, activation=activations[l], init='he_normal'))
          model.add(Dense(1, activation='linear', init='he_normal'))

          # Compiling model
          model.compile(loss='mean_squared_error', optimizer=optimizers[m] , metrics=['mean_absolute_percentage_error'])

          history = model.fit(X_train, y_train, epochs=200, batch_size=8, verbose=0)

          scores = model.evaluate(X_test, y_test)
          scores_all[j].append(float("{:.2f}".format(scores[1])))
        print("Расчет модели с кол-вом нейронов во входном слое = ",
              multiplier[k]*24, ',', ' функцией активации - ',
              activations[l], ' и оптимизатором - ', optimizers[m],
              " закончен с результатами:", sep='')
      print(scores_all)
```

Рисунок 48 – Пример кода наилучшей модели

Результат прогнозирования изображён на рисунке А.1

У обучения на одномерных данных есть очень большое преимущество, не нужно собирать данные о погоде, достаточно только данных графиков нагрузок, а значит нет необходимости совмещать данные, и удалять строки с пропущенными значениями, уже соединенной таблицы, что существенно упрощает обработку данных и существенно увеличивает размер тренировочной выборки, что положительно сказывается на результате прогнозирования.

Обучение данной нейронной сети также было проведено с использованием следующих гиперпараметров:

- Оптимизаторы: SGD, RMSprop, Adam;
- Функции для входного и внутренних слоёв: ReLU, Сигмоида, тангенс функция, ELU;
- Для выходного слоя всегда использовалась линейная функция активации (другие не применимы для данной задачи);
- Также варьировалось количество эпох, первоначальная инициализация весов, количество слоёв и нейронов в них, процентное разбиение тренировочной и тестовых выборок.

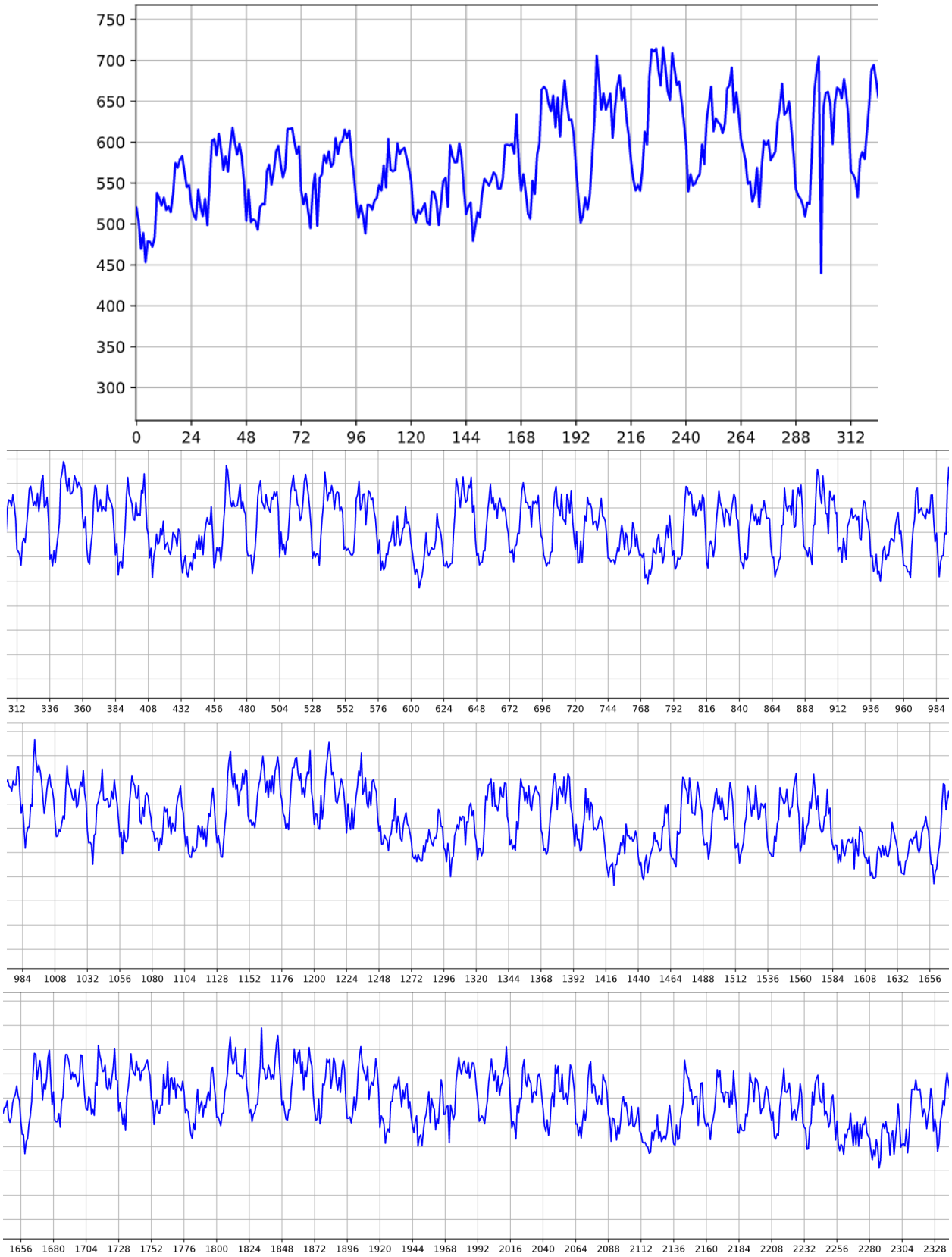
ЗАКЛЮЧЕНИЕ

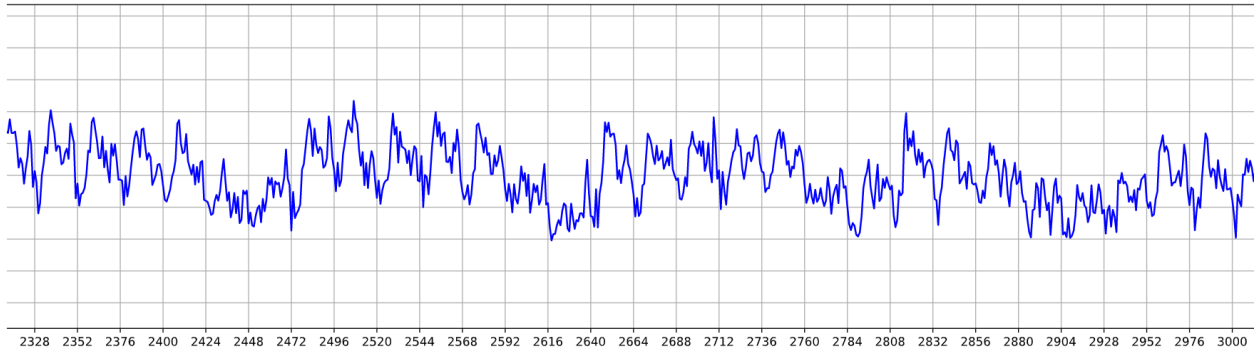
Сравнивая все способы прогнозирования, можно сделать вывод, что нейросети прямого распространения с обучением на одномерных данных хорошо справляются с поставленной задачей, график прогнозируемых величин очень близок к реальному графику нагрузок в рассматриваем узле энергосистемы.

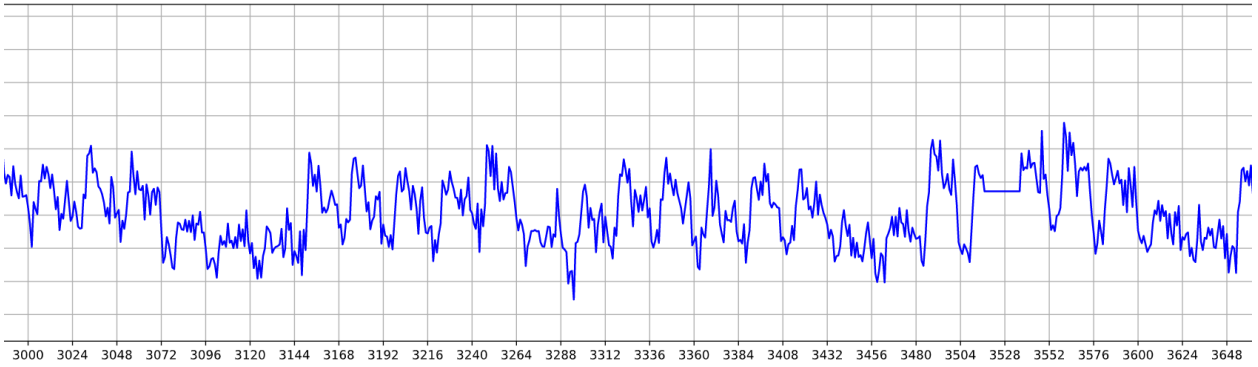
Также стоит обратить внимание на сети RNN архитектуры, однако они более ресурсозатратны в сравнении с сетями прямого распространения и имеют более сложную настройку гиперпараметров и подготовку входных данных.

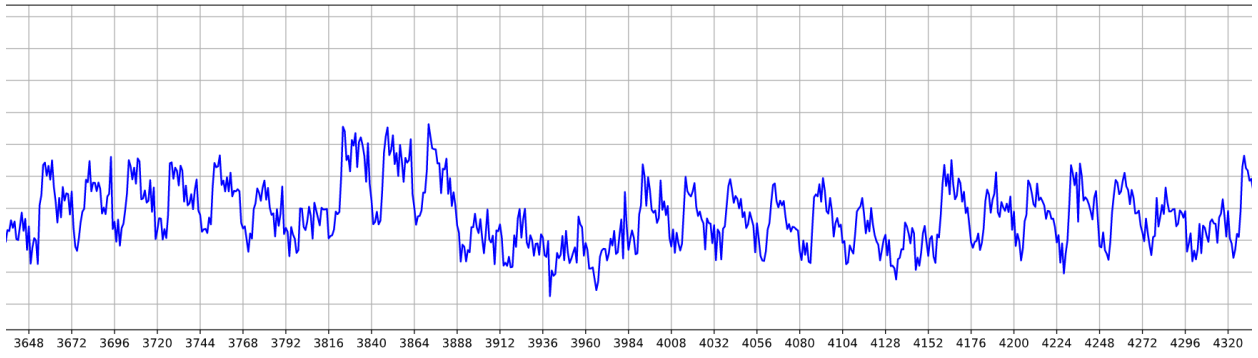
После получения готовой модели нейросети, её полную модель (с сохранением весовых коэффициентов и гиперпараметров) можно записать в виде файла (json формата) и использовать уже в энергосистеме для прогнозирования нагрузок в реальном времени.

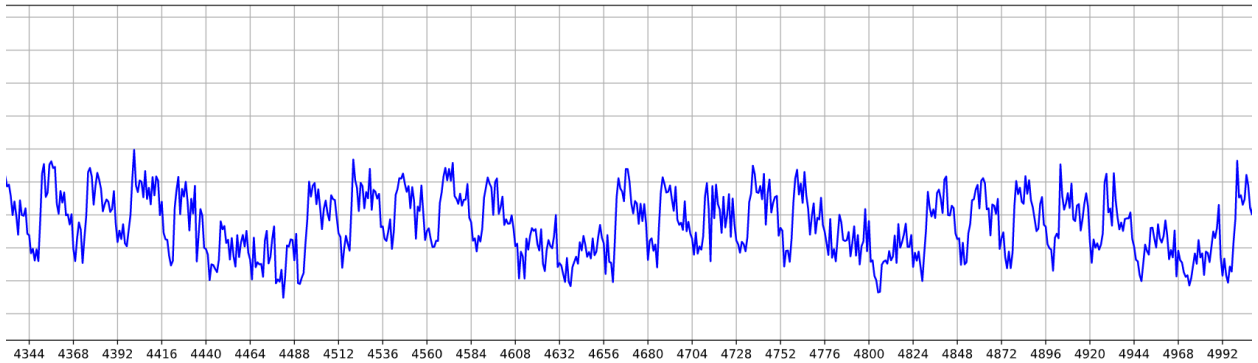
ПИЛОЖЕНИЕ А

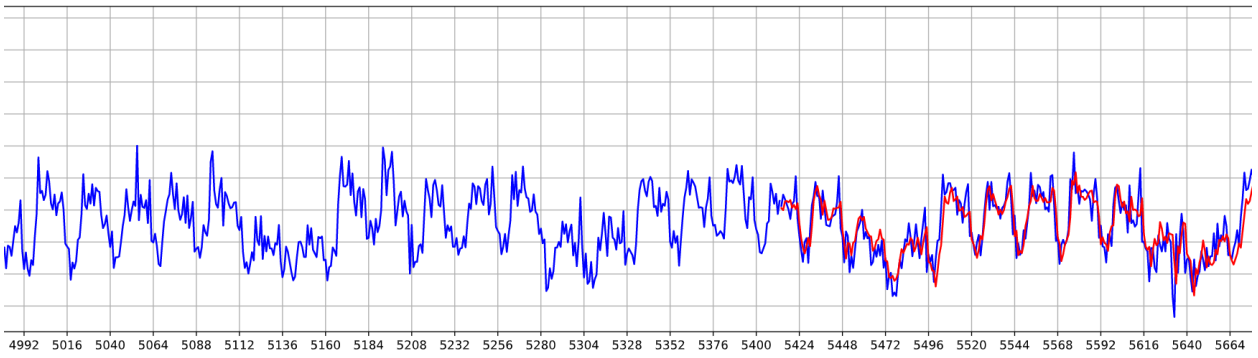

750
700
650
600
550
500
450
400
350
300
0 24 48 72 96 120 144 168 192 216 240 264 288 312
312 336 360 384 408 432 456 480 504 528 552 576 600 624 648 672 696 720 744 768 792 816 840 864 888 912 936 960 984
984 1008 1032 1056 1080 1104 1128 1152 1176 1200 1224 1248 1272 1296 1320 1344 1368 1392 1416 1440 1464 1488 1512 1536 1560 1584 1608 1632 1656
1656 1680 1704 1728 1752 1776 1800 1824 1848 1872 1896 1920 1944 1968 1992 2016 2040 2064 2088 2112 2136 2160 2184 2208 2232 2256 2280 2304 2328


2328 2352 2376 2400 2424 2448 2472 2496 2520 2544 2568 2592 2616 2640 2664 2688 2712 2736 2760 2784 2808 2832 2856 2880 2904 2928 2952 2976 3000


3000 3024 3048 3072 3096 3120 3144 3168 3192 3216 3240 3264 3288 3312 3336 3360 3384 3408 3432 3456 3480 3504 3528 3552 3576 3600 3624 3648


3648 3672 3696 3720 3744 3768 3792 3816 3840 3864 3888 3912 3936 3960 3984 4008 4032 4056 4080 4104 4128 4152 4176 4200 4224 4248 4272 4296 4320


4344 4368 4392 4416 4440 4464 4488 4512 4536 4560 4584 4608 4632 4656 4680 4704 4728 4752 4776 4800 4824 4848 4872 4896 4920 4944 4968 4992


4992 5016 5040 5064 5088 5112 5136 5160 5184 5208 5232 5256 5280 5304 5328 5352 5376 5400 5424 5448 5472 5496 5520 5544 5568 5592 5616 5640 5664

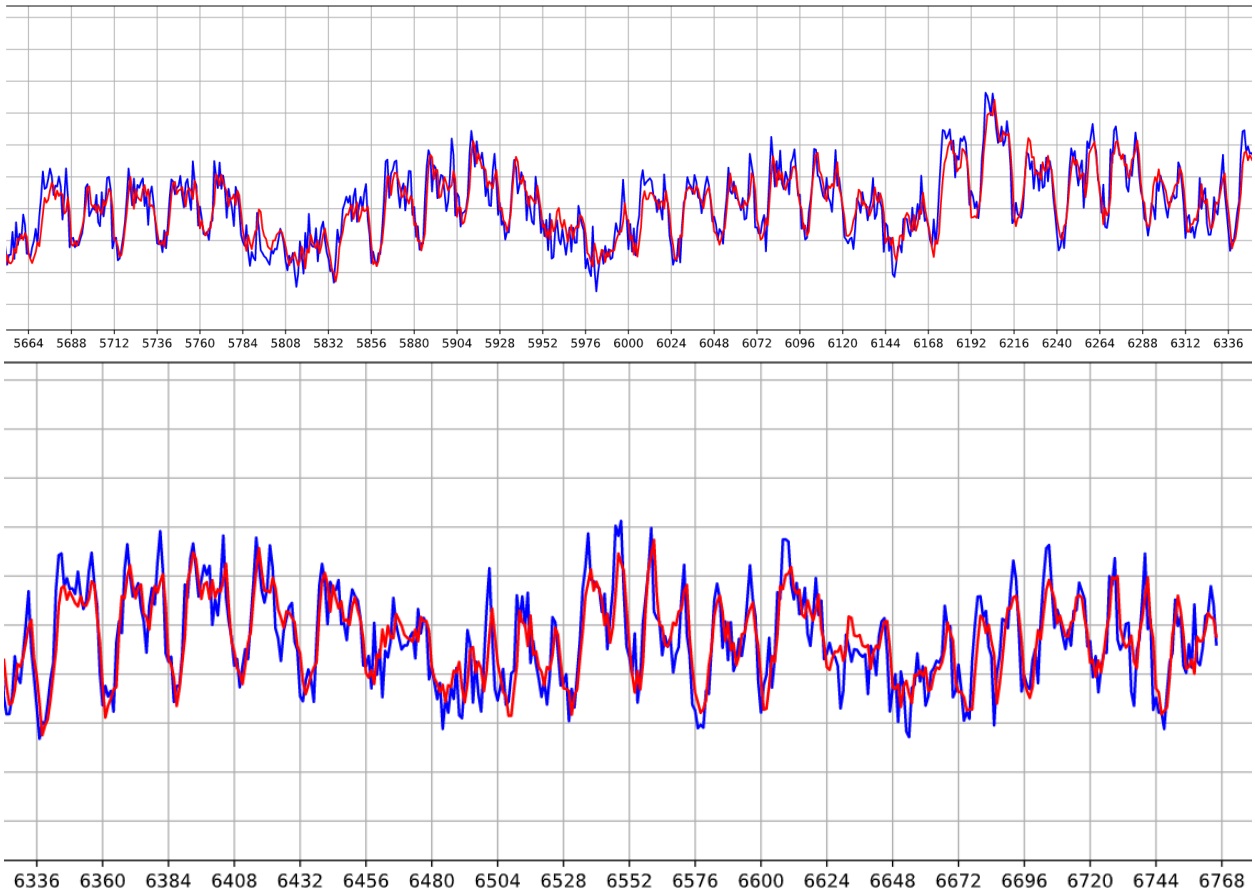


Рисунок А.1 – График нагрузок за год (красный цвет – спрогнозированные величины на тестовой выборке)

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Введение в машинное обучение. – https://habr.com/ru/post/448892/
2. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество. – https://habr.com/ru/post/348000/
3. Методы оптимизации нейронных сетей. – https://habr.com/ru/post/318970/
4. Getting Started with Deep Learning. – https://www.kdnuggets.com/2017/03/getting-started-deep-learning.html/
5. Быстрый старт: обзор основных Deep Learning фреймворков. – https://habr.com/ru/company/newprolab/blog/325818/
6. Developer guides. – https://keras.io/guides/
7. TensorFlow guide. – https://www.tensorflow.org/guide/