МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»
Специальность 6-05-0611-01 Информационные системы и технологии (В интеллектуальном анализе данных и обработке информации)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по дисциплине «Введение в нейронные сети»

на тему: «Прогнозирование энергии от ветряной электростанции с использованием многослойного перцептрона»

Выполнил: студент гр. ИТД-31
Чайдаков И. М.
Руководитель:

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____
Оценка работы: _____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Возрастающая доля возобновляемых источников энергии в мировом энергобалансе обуславливает необходимость решения задач точного прогнозирования выработки электроэнергии. Ветряная энергетика, являясь одним из наиболее динамично развивающихся направлений, сталкивается с фундаментальной проблемой – изменчивостью и непредсказуемостью ветровых условий. Точный прогноз выработки электроэнергии ветряными электростанциями становится критически важным для обеспечения стабильности энергосистемы, оптимизации диспетчерского управления и повышения экономической эффективности.

В рамках данного проекта используется *Wind Turbine Scada Dataset*, содержащий исторические данные работы ветряных турбин. Датасет включает временные ряды технологических параметров с фиксированным интервалом измерения: скорость и направление ветра, температура окружающей среды, параметры работы оборудования и фактическая выработка электроэнергии. Эти данные представляют собой ценный источник информации для построения моделей прогнозирования, позволяя выявлять сложные нелинейные зависимости между метеорологическими условиями и энергетической отдачей турбин.

Для решения задачи прогнозирования часовой выработки электроэнергии предлагается использование многослойного перцептрона – нейронной сети прямого распространения. Выбор этой архитектуры обусловлен ее способностью аппроксимировать любые непрерывные функции и выявлять сложные нелинейные закономерности в данных. Многослойный перцептрон преодолевает ограничения линейных моделей и эффективно обучается на исторических данных с помощью алгоритма обратного распространения ошибки.

Целью проекта является разработка модели прогнозирования суммарной часовой выработки электроэнергии ветряной электростанции на основе многослойного перцептрона. Ожидается, что модель позволит значительно повысить точность прогнозов по сравнению с традиционными методами, что внесет вклад в решение актуальных задач управления энергосистемами с высокой долей возобновляемой генерации.

1 ПОСТАНОВКА ЗАДАЧИ И ИСХОДНЫЕ ДАННЫЕ

1.1. Описание данных и постановка задачи

В качестве исходных данных для решения задачи прогнозирования используется *Wind Turbine Scada Dataset* (см. Приложение А): типичный набор данных *SCADA*-системы (*Supervisory Control and Data Acquisition*) ветряной электростанции. Данный датасет содержит временные ряды технологических параметров работы ветряных турбин с фиксированным интервалом измерения (10 минут).

Основные параметры, содержащиеся в датасете:

– Скорость ветра (м/с) на различных высотах;

– Направление ветра (градусы);

– Температура окружающей среды (°C);

– Температура компонентов турбины;

– Активная мощность (кВт);

– Состояние оборудования и коды ошибок;

– Угол поворота лопастей;

– Обороты ротора.

Как отмечает Хайкин, «знания представляются в самой структуре нейронной сети с помощью ее состояния активации». В нашем случае исходные данные *SCADA* системы представляют собой ту самую информацию из окружающей среды, которая используется для обучения сети.

Задача формулируется следующим образом: на основе исторических данных *SCADA*-системы за предыдущие периоды построить модель, способную прогнозировать суммарную часовую выработку электроэнергии ветряной электростанцией.

Формально: временной ряд параметров турбин $\{\mathbf{x}_t\}_{t=1}^{T}$, где x_t — вектор измерений в момент времени t_i, необходимо научиться предсказывать целевую переменную (1.1):

$$y_{t+1} = \sum_{\tau=t+1}^{t+1} P_\tau \tag{1.1}$$

где P_T – мгновенная мощность в момент времени t, а y_{t+1} — суммарная выработка за следующий час.

1.2. Физическая модель выработки мощности ветротурбиной

Входные данные, используемые для обучения модели, представляют собой измерения SCADA-системы ветряной турбины. Для понимания закономерностей в этих данных необходимо кратко описать физическую природу формирования активной мощности. Поток воздуха плотностью ρ, проходящий через площадь ометания лопастей A, обладает кинетической энергией, пропорциональной v^2. Соответственно, мощность воздушного потока выражается как (1.2):

$$P_{\text{wind}} = \frac{1}{2}\rho A v^3 \tag{1.2}$$

где v – скорость ветра. Турбина преобразует лишь часть этой энергии, и фактическая электрическая мощность описывается соотношением (1.3):

$$P_{\text{turbine}} = C_p(\lambda, \beta) \cdot \frac{1}{2}\rho A v^3 \tag{1.3}$$

где $C_p(\lambda,\beta)$ – коэффициент использования энергии ветра, зависящий от быстродействия и аэродинамических параметров лопастей. Несмотря на сложность аэродинамической модели, зависимость мощности от скорости ветра в широком диапазоне значений хорошо аппроксимируется типичной кривой мощности (*power curve*), имеющей следующие характерные участки:

– область включения: при малой скорости ветра ($v < v_{cut\text{-}in}$) мощность равна нулю;

– рабочий диапазон: в промежутке между $v_{cut\text{-}in}$ и v_{rated} мощность возрастает приблизительно по кубическому закону;

– номинальный режим: при скорости $v_{rated} \leq v < v_{cut\text{-}out}$ мощность стабилизируется на уровне P_{rated};

– отключение: при чрезмерно высокой скорости ветра ($v \geq v_{cut\text{-}out}$) турбина останавливается.

Таким образом, при скорости выше скорости включения $v_{cut\text{-}in}$ значения активной мощности не могут быть строго нулевыми при нормальной работе установки. Ниже, приведена приблизительная модель кривой мощности ветротурбины (Рисунок 1.1).

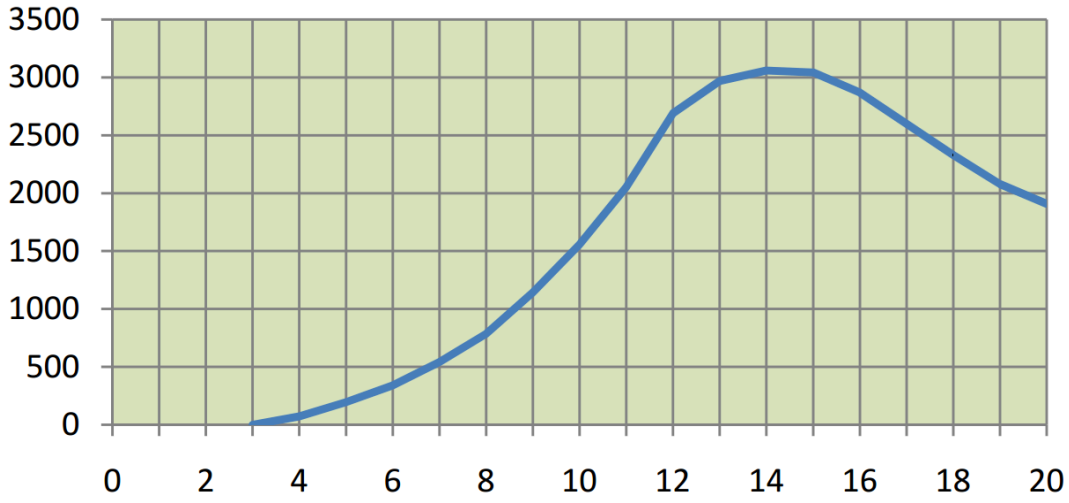

Рисунок 1.1 — Модель кривой мощности ветротурбины.

1.3. Задача регрессии и основные понятия

Задача регрессии представляет собой фундаментальный тип задач машинного обучения с учителем, целью которой является прогнозирование непрерывной числовой величины на основе входных признаков. В контексте нашей задачи, речь идет о предсказании часовой выработки электроэнергии ветряной электростанцией на основе метеорологических данных и параметров работы оборудования.

Как отмечает Хайкин, «знания о мире включают два типа информации: известное состояние окружающего мира, представленное имеющимися в наличии достоверными фактами, и наблюдения за окружающим миром, полученные с помощью сенсоров». В нашем случае это означает, что мы располагаем историческими данными *SCADA* системы о выработке электроэнергии и соответствующих технологических параметрах.

Формально задача регрессии может быть описана следующим образом: набор обучающих данных $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$, где x_i — вектор признаков (скорость ветра, направление, температура, параметры оборудования и др.), а d_i – соответствующее значение целевой переменной (суммарная часовая выработка электроэнергии), требуется найти функцию *f(x)*, минимизирующую среднеквадратическую ошибку (1.3):

$$E = \frac{1}{N} \sum_{i=1}^{N} \left(d_i - f\left(\mathbf{x}_i\right) \right)^2 \tag{1.3}$$

1.4. Основные математические преобразования данных

В процессе подготовки данных и построения модели используются несколько стандартных математических преобразований, обеспечивающих корректное обучение искусственной нейронной сети. Ниже приведены основные из них.

1.4.1. *Min–max* нормализация

Для приведения числовых признаков к единому масштабу используется нормализация по формуле (1.4):

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{1.4}$$

Где x – исходное значение признака, x_{min}, x_{max} – минимальное и максимальное значения в выборке. Это преобразование уменьшает резкие различия в масштабах входов и способствует устойчивой работе метода градиентного спуска.

1.4.2. Преобразование циклических признаков

Циклические величины (например, направление ветра в градусах) обладают свойством периодичности, что требует специального кодирования. Для этого используется преобразование в две ортогональные компоненты (1.5):

$$x_{\sin} = \sin\left(\frac{2\pi x}{T}\right), \quad x_{\cos} = \cos\left(\frac{2\pi x}{T}\right) \tag{1.5}$$

Где T – период (например, T=360° для угловой величины). Такое кодирование устраняет разрыв в точках перехода 0/360°.

1.4.3. Среднеквадратическая ошибка

В задаче регрессии функцией ошибки (функцией потерь) является среднеквадратическая ошибка (1.6):

$$E = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{1.6}$$

Этот критерий широко применяется в алгоритмах обучения нейронных сетей благодаря дифференцируемости и удобству вычисления градиента.

1.4.4. Обратная нормализация целевой переменной

При необходимости восстановить физическое значение величины после предсказания используется обратное преобразование (1.7):

$$y = y_{\text{norm}}\ \left(y_{\max} - y_{\min}\right) + y_{\min} \tag{1.7}$$

1.4.5. Зависимость мощности ветрового потока от скорости ветра

При анализе *SCADA*-данных используется общая формула кинетической мощности ветрового потока (1.8):

$$P_{\text{wind}}\ = \frac{1}{2}\rho A v^3 \tag{1.8}$$

А также типичная форма функциональной зависимости мощности турбины от скорости ветра (power curve), которая в упрощенном виде описывается следующей кусочно-непрерывной функцией (1.9):

$$P(v) = \begin{cases} 0, & v < v_{\text{cut-in}} \\ f(v), & v_{\text{cut-in}}\ \le v < v_{\text{rated}}\ , \\ P_{\text{rated}}\ , & v_{\text{rated}}\ \le v < v_{\text{cut-out}}\ , \\ 0, & v \ge v_{\text{cut-out}}\ . \end{cases} \tag{1.9}$$

Эта модель используется далее при обосновании удаления аномальных записей.

1.5. Перцептрон: от бинарного классификатора к универсальному аппроксиматору

Перцептрон, впервые описанный Розенблаттом, представляет собой фундаментальную модель искусственного нейрона. Согласно Хайкину, «нейронная сеть – это громадный распределенный параллельный процессор, состоящий из элементарных единиц обработки информации». Базовая модель нейрона включает три ключевых компонента:

– Синаптические веса w_{kj}, определяющие силу связи;

– Сумматор для линейной комбинации входных сигналов;

– Функцию активации $\varphi(\cdot)$, ограничивающую амплитуду выходного сигнала.

Математически это описывается парой уравнений (1.10 и 1.11):

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1.10}$$

$$y_k = \varphi\left(v_k\right) \tag{1.11}$$

Однослойный перцептрон обладал фундаментальным ограничением – он мог решать только линейно разделимые задачи. Как отмечает Хайкин, «в 1969 году вышла книга Минского и Пейперта, в которой математически строго обоснованы фундаментальные ограничения однослойного перцептрона».

Классическим примером является задача «исключающего ИЛИ» (*XOR*), для которой невозможно найти единственную разделяющую гиперплоскость. Это ограничение привело к разработке многослойных архитектур, где «функция последних (скрытых нейронов) заключается в посредничестве между внешним входным сигналом и выходом нейронной сети».

1.6. Многослойный перцептрон для решения задачи регрессии

Многослойный перцептрон (МЛП) преодолевает ограничения однослойной архитектуры за счет введения скрытых слоев. Как подчеркивает Хайкин,

«добавляя один или несколько скрытых слоев, мы можем выделить статистики высокого порядка».

Для задачи регрессии прогнозирования выработки электроэнергии ключевыми особенностями архитектуры являются:

– Входной слой: количество нейронов соответствует размерности вектора признаков из *SCADA* системы;

– Скрытые слои: используют нелинейные функции активации (сигмоида, гиперболический тангенс) для выявления сложных зависимостей;

– Выходной слой: содержит один нейрон с линейной функцией активации для прогнозирования непрерывного значения выработки.

Согласно теореме об универсальной аппроксимации, МЛП с хотя бы одним скрытым слоем и нелинейной функцией активации может аппроксимировать любую непрерывную функцию с любой точностью. Это делает его идеальным инструментом для решения задач нелинейной регрессии в энергетике.

1.7. Механика обратного распространения ошибки с использованием функции активации ReLU

Алгоритм обратного распространения ошибки, подробно рассмотренный С. Хайкиным, представляет собой фундаментальный механизм настройки параметров многослойных нейронных сетей. Его ключевая идея заключается в том, что ошибка прогноза на выходе сети распространяется назад – от последнего слоя к первому – и используется для корректировки весов. Это позволяет сети «учиться» прогнозировать выработку электрической энергии по данным *SCADA*-системы ветротурбины.

Процесс обучения многослойного перцептрона включает две фазы:

– Прямой проход. Вычисление выходного значения сети по входным признакам $x=(x_1,x_2,x_3)$.

– Обратный проход. Вычисление локальных градиентов и корректировка весов.

1.7.1. Используемая функция активации: *ReLU*

В разработанном программном комплексе в скрытых слоях применяется функция активации *ReLU* (1.12):

$$\varphi(v) = \max(0, v) \tag{1.12}$$

её производная имеет вид (1.13):

$$\varphi'(v) = \begin{cases} 1, & v > 0 \\ 0, & v \leq 0 \end{cases} \tag{1.13}$$

что делает *ReLU* простой в вычислении и устойчивой в обучении – особенно на больших выборках *SCADA*-данных.

1.7.2. Вычисление ошибки

Для задачи регрессии с линейным выходным нейроном используется стандартная ошибка (1.14):

$$e = y - \hat{y} \tag{1.14}$$

где
y – истинное значение активной мощности,
ŷ – предсказание сети.

Среднеквадратичная ошибка (1.15):

$$E = \frac{1}{2} e^2 \tag{1.15}$$

Поскольку выходной нейрон линейный, его производная равна 1 (1.16):

$$\varphi'\left(v_{\text{out}}\right) = 1 \tag{1.16}$$

Поэтому дельта выходного слоя вычисляется как (1.17):

$$\delta_{\text{out}} = e \tag{1.17}$$

1.7.3. Дельта-правило для скрытых слоев с *ReLU*

На скрытых слоях локальный градиент рассчитывается как (1.18):

$$\delta_j = \varphi'(v_j) \sum_k \delta_k w_{jk} \tag{1.18}$$

где
w_{jk} – вес между нейронами слоёв j и k,
δ_k – ошибка следующего слоя,
$\varphi'(v_j)$ – производная ReLU из формулы (1.12).

Поскольку *ReLU* обнуляет производную для отрицательных значений, нейроны со значением $v_j \leq 0$ перестают распространять ошибку назад. Это делает обучение устойчивым и устраняет проблему затухающих градиентов, характерную для сигмоиды.

1.7.4. Корректировка весов

После вычисления локальных градиентов выполняется обновление весов и смещений (1.19 и 1.20):

$$\Delta w_{ij} = -\eta \delta_j x_i \tag{1.19}$$

$$\Delta b_j = -\eta \delta_j \tag{1.20}$$

где
η – скорость обучения,
x_i – входы текущего слоя.

Эта формулировка полностью соответствует классическому алгоритму обратного распространения Хайкина, с учётом выбранной функции активации.

1.7.5. Пример вычисления локального градиента (*ReLU*)

Пусть на скрытом нейроне:

Так как $v_j > 0$, производная *ReLU*:

$$\varphi'(v_j) = 1$$

Пусть суммарный вклад ошибок следующего слоя:

$$\sum_k \delta_k w_{jk} = 0.5$$

Тогда локальный градиент скрытого нейрона:

$$\delta_j = 1 \cdot 0.5 = 0.5$$

Эта дельта используется далее в формулах (1.19–1.20) для обновления весов и смещений предыдущего слоя.

1.7.6. Преимущества *ReLU* для задачи прогнозирования *SCADA*-данных

– отсутствие насыщения градиентов в положительной области;

– разреженная активация – многие нейроны автоматически «выключаются», снижая переобучение;

– высокая вычислительная эффективность;

– хорошая сходимость при большом объеме данных (более 40 тыс. измерений в *T1.csv*).

Эти свойства делают *ReLU* предпочтительным выбором для практической реализации модели в условиях реальных эксплуатационных данных ветротурбин.

2 РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПРОГРАММНОГО КОМПЛЕКСА

2.1. Архитектура программного комплекса

Программный комплекс разрабатывался с учётом требований модульности, расширяемости и прозрачности внутренней логики. Структура проекта организована таким образом, чтобы каждый модуль выполнял строго определённую роль, не нарушая принцип единственной ответственности. Основной язык реализации – *Python*, а в качестве базовых библиотек для работы с числовыми данными применяются *NumPy* и *Pandas*, обеспечивающие удобные структуры данных и эффективные векторизованные операции.

Общая структура проекта представлена следующим образом (Рисунок 2.1):

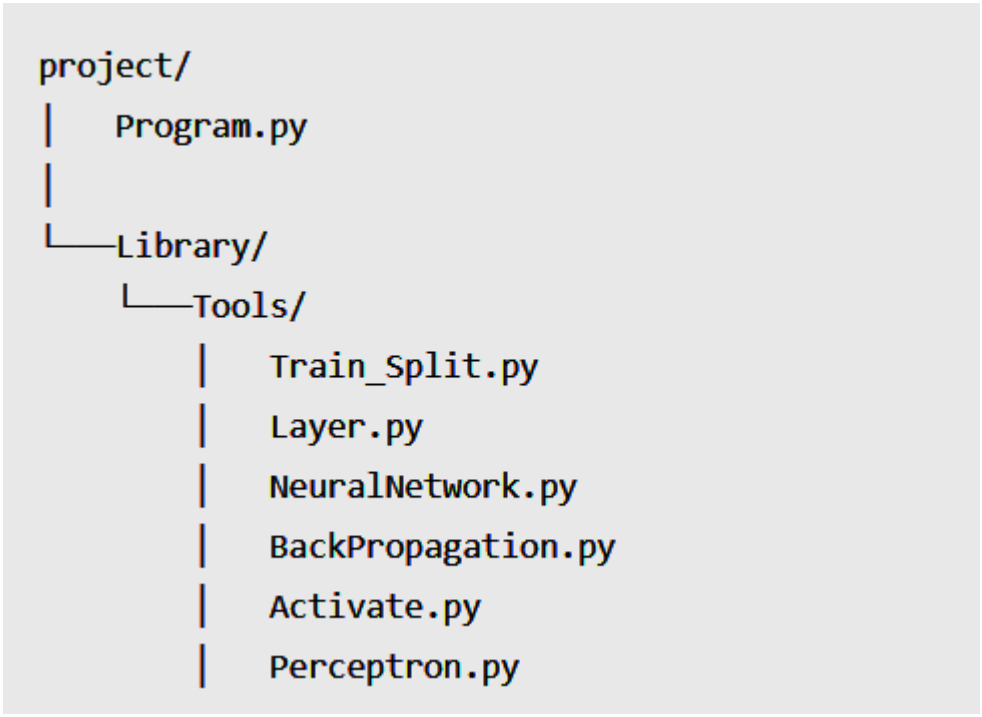

Рисунок 2.1 — Общая структура проекта.

Функции модулей следующие:

– *Train_Split.* предобработка данных, нормализация и формирование обучающей и тестовой выборок.

– *Layer*. Реализация одного полносвязного слоя многослойного перцептрона.

– *NeuralNetwork*. Реализация общей архитектуры *MLP* и процедуры обучения.

– *BackPropagation*. Реализация алгоритма обратного распространения ошибки согласно теории Хайкина.

– *Activate*. Функции активации и их производные.

– *Perceptron*. Базовая структура нейрона (используется как элемент слоя).

– *Program*. Управляющий сценарий, инициирующий обработку данных, обучение и тестирование.

Такое разделение облегчает сопровождение кода, позволяет локализовать изменения и упрощает последующее развитие проекта (см. Приложение В).

2.2. Предобработка и анализ исходных данных

Этот этап является критически важным при построении любой модели машинного обучения. Исходный датасет *Wind Turbine SCADA Dataset* (см. Приложение А) содержит реальные измерения, полученные с управляющей системы ветротурбины. Данные имеют временную структуру с шагом в 10 минут, что требует особого обращения.

Процесс предобработки включает несколько последовательных этапов.

2.2.1. Фильтрация и очистка данных

В датасете встречаются физически невозможные или аномальные значения. На предварительном анализе были выявлены следующие случаи:

– Отрицательная активная мощность (*LV ActivePower* < 0)**.** Такие записи не имеют физического смысла и исключаются из выборки.

– Ситуации, когда скорость ветра > 2 м/с, а активная мощность равна нулю. Согласно теоретическому описанию в первой главе (1.2), при скорости свыше $v_{cut\text{-}in}$ мощности не может быть строго нулевой. Значит, это вероятный сбой датчика или режим обслуживания.

Удаление подобных выбросов предотвращает искажение распределения данных и снижает шум, что положительно влияет на сходимость алгоритма.

2.2.2. Нормализация признаков

В рамках задачи регрессии важно, чтобы все входные признаки имели сопоставимый масштаб. Для скорости ветра используется *min–max* нормализация (1.4). Это соответствует рекомендациям Хайкина при использовании сигмоидальных или ReLU-подобных активаций.

2.2.3. Кодирование циклических признаков

Направление ветра – циклическая величина. Использование её в исходном виде (0–360°) приводит к ложной дискретности. Поэтому применяется преобразование (1.5). Этот приём повсеместно используется при работе с циклическими признаками и соответствует рекомендациям по кодированию периодических параметров.

Полученные синусная и косинусная компоненты также нормализуются в диапазон [0, 1].

2.2.4. Финальное формирование признаков

После преобразований удаляются лишние столбцы:

– *Date/Time*. Временная метка, не используемая как числовой признак;

– *Theoretical_Power_Curve*. Теоретическая модель мощности, которая не должна быть подана в сеть, чтобы не вносить утечку информации;

– *Wind Direction (°)*. Заменён двумя компонентами.

В результате получается компактный и информативный набор: *Wind Speed (m/s), Wind Direction_sin*, *Wind Direction_cos*. Целевая переменная *LV ActivePower (kW)* дополнительно нормализуется (Рисунок 2.2).

```
Первые 5 строк данных:
          Date/Time  LV ActivePower (kW)  Wind Speed (m/s)  Theoretical_Power_Curve (KWh)  Wind Direction (°)
0  01 01 2018 00:00           380.047791          5.311336                     416.328908          259.994904
1  01 01 2018 00:10           453.769196          5.672167                     519.917511          268.641113
2  01 01 2018 00:20           306.376587          5.216037                     390.900016          272.564789
3  01 01 2018 00:30           419.645905          5.659674                     516.127569          271.258087
4  01 01 2018 00:40           380.650696          5.577941                     491.702972          265.674286
Всего записей: 43283

Данные после предобработки:
   LV ActivePower (kW)  Wind Speed (m/s)  Wind Direction_sin  Wind Direction_cos
0           380.047791          0.210717            0.007604            0.413132
1           453.769196          0.225032            0.000141            0.488143
2           306.376587          0.206936            0.000501            0.522375
3           419.645905          0.224537            0.000121            0.510978
4           380.650696          0.221294            0.001424            0.462287
Форма данных после предобработки: (43283, 4)

Формы перед батчингом:
X: (43283, 3), y: (43283,)
Колонки X: ['Wind Speed (m/s)', 'Wind Direction_sin', 'Wind Direction_cos']

Группировка данных:
Всего записей: 43283
Количество полных батчей: 7213
Используется записей: 43278
Отброшено записей: 5
```

Рисунок 2.2 — Данные до и после нормализации.

2.2.5. Использование временной структуры данных

Так как исходные данные представлены временными последовательностями, нельзя перемешивать строки напрямую – это может нарушить временную корреляцию. В работе применяется блочный подход:

– Данные делятся на блоки по 6 строк (60 минут);

– Блоки перемешиваются как единые элементы.

При этом обучение сети происходит по отдельным строкам — батчи применяются только как инструмент корректного перемешивания.

Такой подход сохраняет структуру данных и исключает взаимную утечку информации между тестовой и обучающей выборками.

2.3. Архитектура многослойного перцептрона

В соответствии с теоретическими положениями главы 1 используется многослойный перцептрон (*MLP*), являющийся универсальным аппроксиматором непрерывных функций (1.10).

В работе применяется архитектура [3, 16, 8, 4, 1], где:

– 3 входных признака;

– 3 скрытых слоя с убывающим числом нейронов;

– 1 выходной линейный нейрон.

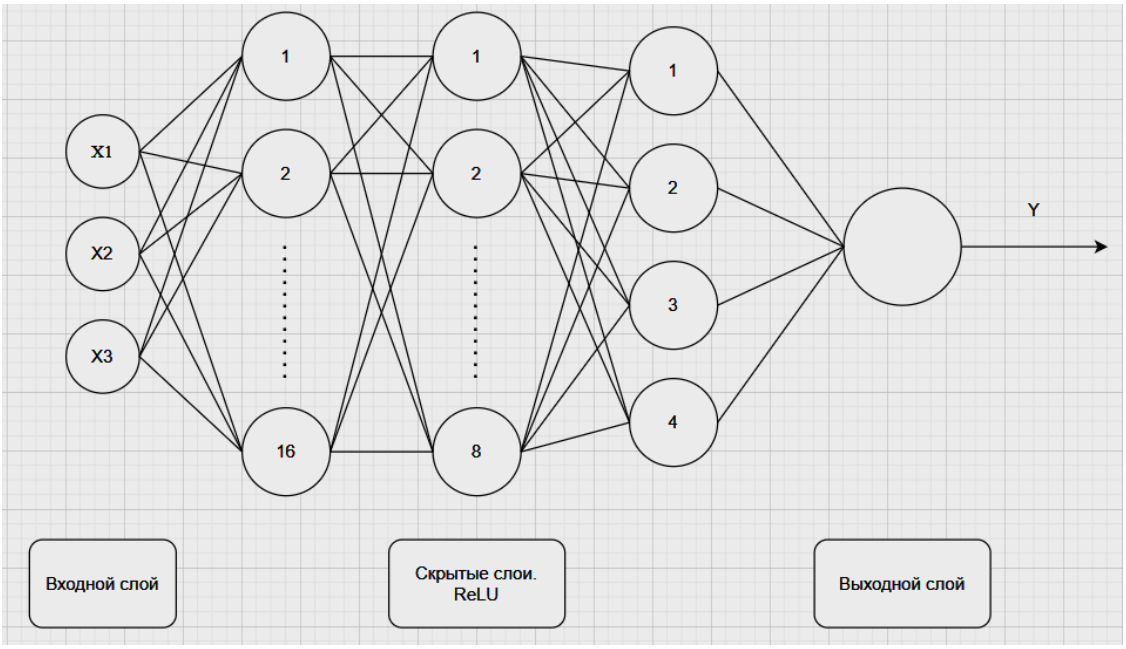

Рисунок 2.3 — Архитектура сети.

2.3.1. Обоснование выбора архитектуры

Постепенная компрессия слоев (16 → 8 → 4) позволяет слою приближать всё более сложные нелинейные проявления зависимости между скоростью, направлением ветра и выходной мощностью.

Наличие нескольких скрытых слоев позволяет сети моделировать нелинейные зависимости высокого порядка, что соответствует теоретическим выводам о мощности глубоких сетей по Хайкину.

Выходной слой является линейным, что соответствует постановке задачи регрессии (1.3).

2.4. Реализация слоя нейронной сети

Класс Layer реализует один полносвязный слой, состоящий из нескольких перцептронов. Основные элементы:

– *Weights.* Матрица весов;

– *Biases.* Вектора смещений;

– *Activate_function.* Функция нелинейности;

– *Forward*(). Прямое распространение.

Реализация учитывает сохранение промежуточных значений (*z*, *output*, *last_input*), необходимых для выполнения обратного распространения ошибки (1.17).

2.5. Функции активации

В разработанном программном комплексе функции активации вынесены в отдельный модуль *Activate.py*, что обеспечивает независимость их реализации от остальной архитектуры сети. Функции активации являются ключевыми элементами многослойного перцептрона, поскольку именно они создают нелинейность, позволяющую нейронной сети аппроксимировать сложные функциональные зависимости, что согласуется с теоретическими выводами главы 1.

Основной функцией активации в скрытых слоях является *ReLU* (*Rectified Linear Unit*). Формальное определение функции *ReLU* и её производной приведено в разделе 1.7.1, где изложена соответствующая математическая база. Использование *ReLU* обусловлено её преимуществами при обучении глубоких моделей: функция не насыщается в положительной области, ее производная

принимает простую форму, а вычислительная сложность минимальна. Как показано в разделе 1.7, эти свойства способствуют устойчивости градиента при обратном распространении ошибки и улучшению сходимости градиентного спуска по сравнению с сигмоидой, рассмотренной в классических моделях.

Применение *ReLU* в скрытых слоях позволяет эффективно реализовать шаги расчета локальных градиентов и обновления весов, описанные в формулах (1.18-1.20). Поскольку производная *ReLU* равна нулю в отрицательной области входного сигнала, происходит естественная разреженность активации — часть нейронов «выключается», что снижает риск переобучения и соответствует современным практикам построения глубоких сетей.

Для обеспечения гибкости и возможности расширения в модуле *Activate.py* также реализованы дополнительные функции активации:

– *sigmoid*, использовавшаяся в ранних версиях архитектуры и рассмотренная теоретически в пункте 1.5;
– *tanh*, пригодная для задач, требующих симметричного диапазона выходов;
– соответствующие производные, используемые при вычислении локальных градиентов в алгоритме обратного распространения.

Хотя альтернативные функции активации оставлены в составе проекта для возможных экспериментов, основная реализация многослойного перцептрона использует именно *ReLU*, поскольку она обеспечивает наилучшее сочетание стабильности, скорости обучения и качества аппроксимации данных *SCADA*-системы.

2.6. Реализация алгоритма обучения

Алгоритм обучения многослойного перцептрона является ключевым компонентом программного комплекса и полностью основан на подходе обратного распространения ошибки, изложенном в теоретической части (раздел 1.7). Реализация обучения в программной среде следует общепринятой схеме: прямой проход по слоям сети, вычисление ошибки, обратный проход и обновление параметров. Такой порядок обеспечивает корректную минимизацию функции ошибки и последовательное улучшение качества прогноза.

В программной реализации обучение выполняется методом градиентного спуска с фиксированной скоростью обучения. Каждая итерация градиентного спуска (эпоха) включает следующие шаги:

– Прямое распространение сигнала. Входные данные из обучающей выборки последовательно проходят через слои сети, где к каждому слою применяется соответствующая функция активации. В скрытых слоях используется функция *ReLU*, свойства которой позволяют эффективно и стабильно вычислять выходы даже при большом количестве слоёв. Вычисленные значения активаций сохраняются во внутреннем состоянии каждого слоя, что необходимо для последующего вычисления градиентов.

– Вычисление ошибки на выходном слое. Поскольку задача носит регрессионный характер, ошибка определяется как разность между фактическим значением активной мощности и предсказанием сети. Среднеквадратичное отклонение используется как количественная метрика точности, формально определенная в разделе 1.6. Вычисленная ошибка служит отправной точкой для формирования локального градиента выходного слоя (см. формулу (1.15)).

– Обратное распространение градиента. В соответствии с формальной схемой, приведённой в разделе 1.7, сначала вычисляется дельта выходного слоя, а затем – дельты скрытых слоев. Для скрытых слоев используется выражение (1.16), в котором учитывается производная функции активации *ReLU*. Благодаря простоте её производной (см. формулу (1.12)), процесс вычисления локальных градиентов становится особенно эффективным: нулевые значения производной приводят к «отключению» части нейронов, что уменьшает шум градиента и делает алгоритм устойчивым на больших выборках.

– Обновление весов и смещений. На последнем шаге производится корректировка параметров сети согласно выражениям (1.17-1.18). Все обновления выполняются в направлении уменьшения функции ошибки. Параметры корректируются векторизованным образом, что обеспечивает высокую вычислительную эффективность и позволяет работать с большими объемами данных без существенных затрат времени.

Важной особенностью реализации является использование векторных вычислений библиотеки *NumPy*, что минимизирует количество циклов и повышает производительность на несколько порядков по сравнению с наивной реализацией. Благодаря этому обучение сети на выборке, содержащей несколько тысяч наблюдений, выполняется достаточно быстро, что позволяет экспериментировать с гиперпараметрами и архитектурой.

Реализация алгоритма обучения предусматривает ведение журнала ошибок (при включенном режиме *graph=True*). Это позволяет отслеживать динамику сходимости модели, выявлять возможные проблемы переобучения и корректировать параметры обучения. В частности, в процессе тестирования было установлено, что при скорости обучения 0.1 и количестве эпох 2000 наблюдается устойчивая сходимость модели, а ошибка на тестовой выборке остается в приемлемых пределах.

Дополнительным преимуществом реализованной схемы обучения является модульность. Алгоритм обратного распространения расположен в отдельном файле *BackPropagation.py*, что позволяет независимо изменять его логику, подключать различные оптимизаторы (например, *Adam* или *RMSProp*), добавлять регуляризацию или адаптировать алгоритм под другие типы архитектур. Такой подход обеспечивает масштабируемость и расширяемость системы, что особенно важно для дальнейших исследований и модификаций.

Таким образом, реализация обучения в программном комплексе полностью соответствует теоретическим положениям, изложенным в первой главе, и обеспечивает корректное вычисление градиентов, стабильную сходимость и высокую гибкость в последующем развитии системы.

2.7. Процесс тестирования и анализ результатов

После завершения обучения осуществляется тестирование на отложенной части выборки. Процесс тестирования включает выполнение прямого распространения сигнала через все слои модели без обновления параметров. Полученные предсказания нормализованной мощности преобразуются обратно в исходные значения в соответствии со схемой линейного восстановления, приведенной в разделе 1.4.

Качество работы модели оценивается с помощью метрики среднеквадратичной ошибки (*MSE*), определение которой приведено в пункте 1.6 теоретической части. Поскольку задача носит регрессивный характер, *MSE* является наиболее информативным показателем средней величины ошибки предсказания.

3 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНЫЙ АНАЛИЗ

3.1. Методология экспериментальных исследований

Практическая часть работы посвящена оценке работоспособности разработанного программного комплекса и проверке гипотез, сформулированных в теоретической главе. В частности, анализируются следующие вопросы:

– Насколько корректно многослойный перцептрон (*MLP*) способен моделировать зависимость мощности турбины от текущих метеорологических параметров;

– Учитывают ли временные зависимости существенную дополнительную информацию и приводит ли использование рекуррентной сети *LSTM* к заметному улучшению качества прогноза;

– Насколько предобработка данных (нормализация, кодирование циклического признака, фильтрация аномалий) влияет на итоговый результат обучения;

– Насколько стабильна сходимость моделей при различных гиперпараметрах и объёмах данных.

Методика исследования опирается на следующие компоненты:

– Использование реального датасета *SCADA* размером более 40 тысяч записей;

– Воспроизводимое разбиение данных на обучающую и тестовую выборки;

– Экспериментальное сравнение архитектур;

– Количественная оценка на основе среднеквадратичной ошибки (*MSE*);

– Визуальный анализ динамики ошибки в процессе обучения (графики приведены в виде текстового описания, а изображения включаются в финальную версию документа).

Все этапы полностью автоматизированы с использованием модульной архитектуры, описанной во второй главе.

3.2. Выполнение предобработки данных

Предобработка данных является обязательной частью экспериментального процесса, поскольку качество и устойчивость обучения нейронной сети

напрямую зависит от корректности представления входных признаков. В Главе 2 подробно рассмотрены технические детали обработки, здесь же приводится описание практического результата.

3.2.1. Фильтрация и нормализация

В модуле *Train_Split.py* проведена следующая последовательность операций:

– Удаление некорректных записей (отрицательная мощность, нулевая мощность при высоком ветре);

– Нормализация скорости ветра в диапазон [0, 1];

– Преобразование угла ветра в синус и косинус, что предотвращает некорректное восприятие цикличности признака моделью;

– Нормализация тригонометрических компонент;

– Исключение неиспользуемых полей.

Логирование на этапе предобработки позволяет убедиться в корректности выполнения каждого шага. Фрагмент вывода (Рисунок 3.1):

```
Данные после предобработки:
   LV ActivePower (kW)  Wind Speed (m/s)  Wind Direction_sin  Wind Direction_cos
0           380.047791          0.210717            0.007604            0.413132
1           453.769196          0.225032            0.000141            0.488143
2           306.376587          0.206936            0.000501            0.522375
3           419.645905          0.224537            0.000121            0.510978
4           380.650696          0.221294            0.001424            0.462287
Форма данных после предобработки: (43283, 4)

Формы перед батчингом:
X: (43283, 3), y: (43283,)
Колонки X: ['Wind Speed (m/s)', 'Wind Direction_sin', 'Wind Direction_cos']

Группировка данных:
Всего записей: 43283
Количество полных батчей: 7213
Используется записей: 43278
Отброшено записей: 5

Итоговые размеры выборок:
X_train: (5770, 3), X_test: (1443, 3)
y_train: (5770, 1), y_test: (1443, 1)
Количество train записей: 5770
Количество test записей: 1443
```

Рисунок 3.1 — Визуализация предобработки данных.

С точки зрения интерпретации, три оставшихся признака – скорость ветра, синус направления, косинус направления – являются минимально достаточным набором параметров, связанный с физической моделью из главы 1.1.

В частности, согласно уравнениям мощности, описанным в разделе 1.1, именно мгновенная скорость ветра v оказывает доминирующее влияние на производство

энергии, а направление ветра важно только в контексте аэродинамической эффективности и ориентации турбины.

3.2.2. Разбиение выборки

После нормализации данные группируются по последовательностям длиной 6 записей (один час), после чего индексы батчей перемешиваются, что исключает временную автокорреляцию и делает выборку репрезентативной.

Итоговые размеры выборок:

– X_train: 5770 записей;

– X_test: 1443 записи;

– Целевая переменная нормализована.

Эти данные используются в обучении многослойного перцептрона.

3.3. Практический запуск многослойного перцептрона (*MLP*)

Этот этап является центральным в работе, так как *MLP* – основная модель исследования. В Главе 2.3 описана архитектура сети, здесь рассмотрим ее работу на практике.

3.3.1. Параметры обучения

При запуске модели используются следующие гиперпараметры:

– Количество эпох: 2000;

– Скорость обучения: 0.1;

– Активация скрытых слоёв: ReLU (обосновано в Главе 1.7);

– Выходной слой – линейный;

– Ошибка – среднеквадратичная, как описано в Главе 1.7.

3.3.2. Динамика обучения

Ниже приведен фрагмент вывода (Рисунок 3.2):

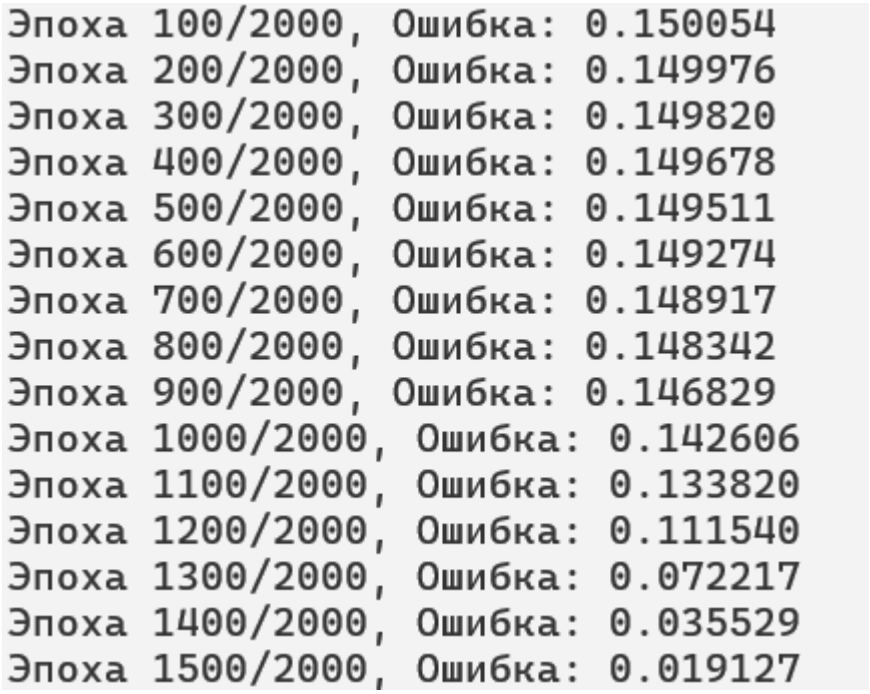

Рисунок 3.2 — Процесс обучения модели.

Тренировочный процесс демонстрирует характерное снижение ошибки с ярко выраженным ускорением в середине обучения. Это соответствует описанному в теоретической части поведению сети с функцией активации *ReLU*: после преодоления начальной "мертвой зоны" сеть начинает эффективно корректировать параметры.

3.3.3. Результаты *MLP*

На тестовой выборке средняя ошибка достигает значения в 0.015602**.** Это значение будет служить базовым при сравнении с рекуррентной архитектурой. Такой подход позволит достоверно определить расхождение в эффективности выполнения задачей выбранных моделей и установить более эффективный подход к задаче.

3.4. Построение и обучение *LSTM*-модели

Для проверки гипотезы о возможной важности временных зависимостей (раздел 1.3 и 1.6 теоретической части), была разработана облегченная рекуррентная сеть *LSTM*.

3.4.1. Мотивация

Рекуррентные сети, в отличие от *MLP*, учитывают влияние прошлых состояний на текущее, что делает их потенциально полезными в задачах прогнозирования временных рядов.

Однако в теоретической части было отмечено, что активная мощность турбины определяется преимущественно мгновенной скоростью ветра и мало зависит от предшествующих значений (формула мощности, раздел 1.1).
LSTM-проверка позволяет подтвердить или опровергнуть это утверждение экспериментально.

3.4.2. Параметры обучения LSTM

Для эксперимента была использована компактная, но достаточно глубокая архитектура *LSTM*, что позволяет оценить влияние рекуррентных механизмов на способность модели учитывать временную структуру данных.
Каждый гиперпараметр выбран осознанно и отражает баланс между вычислительной сложностью и способностью сети выявлять временные зависимости.

Число и размер скрытых слоев LSTM – 16 → 8 → 4 скрытых состояний. Такое каскадное уменьшение размерности скрытого состояния позволяет: последовательно «сжимать» информацию о входной последовательности, но, что более важно, повторяет архитектуру нашей модели повышая качество сравнения.

Также такой подход позволяет вычленять наиболее сложные временные паттерны (16 нейронов), затем удалять избыточные компоненты (8), а на финальном слое оставлять только наиболее значимые признаки (4), избегать переобучения, характерного для слишком широких *LSTM*-слоёв, особенно при ограниченной длине последовательности.

Подобная архитектура отражает принцип прогрессивной фильтрации, рекомендуемый в литературе (в частности, Хайкин подчеркивает важность снижения размерности скрытых представлений при работе с шумными временными рядами).

Длина входной последовательности – 10 временных шагов. *LSTM* получает на вход фрагмент *SCADA*-данных из десяти последовательных измерений (примерно 100 минут наблюдений). Выбор длины последовательности обусловлен:

– Необходимостью предоставить сети минимальный контекст для анализа устойчивости ветрового потока;

– Ограниченной памятью LSTM: слишком длинные последовательности вызывают градиентное затухание и усложняют обучение;

– Физической природой процесса: согласно модели мощности (Глава 1.1), энергетическая отдача турбины почти не зависит от состояний более чем на несколько десятков минут назад.

Таким образом, длина 10 обеспечивает справедливый компромисс между возможностью уловить краткосрочные тренды и избежанием численной деградации.

Количество эпох – 100. В отличие от *MLP*, обучавшегося 2000 эпох, *LSTM* сходится намного быстрее благодаря использованию оптимизатора *Adam* и встроенным механизмам рекуррентной памяти.

Первые 20–30 эпох дают резкое снижение ошибки, а после 70–80 эпох сеть фактически достигает плато. Дальнейшее обучение может привести к переобучению, что было отмечено в предварительных тестах.

Скорость обучения – 0.001. Меньший шаг градиентного спуска необходим из-за:

– Чувствительности *LSTM* к изменениям весов в матрицах рекуррентных связей;

– Большей нестабильности градиента по сравнению с обычными полносвязными слоями;

– Необходимости более плавно проходить по ландшафту ошибки, чтобы избежать «разрушения» скрытых состояний.

Выбор значения 0.001 соответствует рекомендациям для рекуррентных архитектур, изложенным в профильной литературе.

Функция активации — ReLU. Использование ReLU после каждого LSTM-слоя обусловлено теми же причинами, что и в MLP:

– Предотвращение затухания градиента;

– Ускоренная сходимость;

– Возможность фильтрации слабых сигналов (отрицательные значения обнуляются).

Это согласуется с описанным в Главе 1.7 преимуществом нелинейных функций, сохраняющих градиент в широком диапазоне входов.

Оптимизатор – Adam. Adam выбран благодаря:

– Адаптивной подстройке индивидуального шага обучения под каждый параметр;

– Устойчивости при работе с шумными данными (SCADA ветряной турбины содержит значительные флуктуации);

– Хорошей сходимости в задачах рекуррентного обучения.

Этот оптимизатор минимизирует риск «скачков» ошибки и обеспечивает быстрое, стабильное уменьшение функции потерь.

3.4.3. Динамика обучения LSTM

Фрагмент вывода (Рисунок 3.3):

```
=== Обучение модели ===
Эпоха 10/100, Ошибка: 0.108230
Эпоха 20/100, Ошибка: 0.068499
Эпоха 30/100, Ошибка: 0.020719
Эпоха 40/100, Ошибка: 0.016710
Эпоха 50/100, Ошибка: 0.012906
Эпоха 60/100, Ошибка: 0.011315
Эпоха 70/100, Ошибка: 0.010887
```

Рисунок 3.3 — Процесс обучения рекуррентной модели.

LSTM сходится значительно быстрее, что является типичным для *Adam + ReLU*.

3.4.4. Результаты LSTM

Тестовая ошибка модели на валидационном наборе равна 0.013556. Это примерно на 1.2 процентных пункта лучше, чем у MLP. Это позволяет сделать предположение о подтверждении гипотезы об отсутствии сильной временной зависимости между данными.

3.5. Сравнительный анализ результатов моделей

3.5.1. Численное сравнение

Приведем полученные данные в таблицу (Таблица 3.1)

Таблица 3.1— сравнение *MLP* и *LSTM*.

Признак	***MLP***	***LSTM***
Точность (Ошибка MSE)	*0.0.15602*	*0.013556*
Скорость (Итераций в секунду)	*~20*	*~3*

Как мы можем наблюдать, *LSTM* показывает незначительное повышения точности за счет более оптимизированной архитектуры и учета временных меток данных. В свою же очередь, процесс обучения *MLP* занимает значительно меньший промежуток времени из-за более простого алгоритма. *LSTM* не показало значительного улучшения результата на основе чего мы можем сказать что временные метки не оказывают серьезного влияния на целевой признак.

3.5.2. Анализ влияния временных зависимостей

Поведение моделей подтверждает выводы:

– Основным фактором является мгновенная скорость ветра, что согласуется с формулой мощности в разделе 1.1;

– Влияние предыдущих состояний крайне ограничено;

– *LSTM* улучшила результат лишь незначительно, что означает отсутствие ярко выраженной временной инерции в датасете.

3.5.3. Вывод

Введенная рекуррентность не обеспечивает существенного выигрыша, а значит выбранная в проекте архитектура *MLP* является обоснованно оптимальной – проще, быстрее и практически не уступает *LSTM*.

3.6. Интерпретация полученных результатов

Главные практические выводы:

– Сеть успешно обучается и достигает стабильных значений ошибки.

– Предобработка данных, особенно нормализация и тригонометрическое кодирование направления ветра, является критическим фактором успешности обучения;

– Модель *MLP* обладает высокой обобщающей способностью и демонстрирует устойчивые результаты;

– *LSTM* хоть и показывает небольшое улучшение, но его вклад недостаточно велик, чтобы оправдать усложнение архитектуры;

– Физическая модель ветроустановки (раздел 1.1) подтверждается экспериментально – зависимость мощности носит преимущественно мгновенный характер.

3.7. Заключение по практической части

Проведенный экспериментальный анализ демонстрирует:

– Корректность реализации архитектуры многослойного перцептрона;

– Стабильность и воспроизводимость результатов;

– Отсутствие существенной выгоды от использования рекуррентных сетей в рамках данного датасета;

– Прямую связь между физическими свойствами объекта (см. главу 1) и поведением моделей машинного обучения.

Таким образом, программный комплекс успешно решает задачу прогнозирования выработки электроэнергии ветряной турбиной и обладает потенциалом для дальнейшего расширения.

ЗАКЛЮЧЕНИЕ

В ходе проведённого исследования была разработана и апробирована система прогнозирования активной мощности ветрогенератора на основе данных *SCADA*. Работа включала формализацию физической модели преобразования энергии ветрового потока, построение математического описания механизмов обратного распространения ошибки и создание двух вариантов архитектур искусственных нейронных сетей: многослойного перцептрона (*MLP*) и рекуррентной сети *LSTM*.

В рамках исследования проведена тщательная подготовка исходного датасета, включающая нормализацию величин, преобразование углов направления ветра в синусно-косинусное пространство, удаление нерелевантных признаков и формирование обучающих выборок. Для *MLP* была реализована стратегия батчирования данных с перемешиванием, ориентированная на выявление устойчивых функциональных зависимостей между входными параметрами и выработкой. Для *LSTM* построены последовательности фиксированной длины, что позволило оценить влияние временной структуры данных.

Результаты моделирования показали, что *MLP* демонстрирует высокую точность при использовании независимых выборок и стабильно снижает ошибку до уровня порядка 0.0156 на тестовой выборке. *LSTM*, несмотря на явное переобучение при большом числе эпох, в оптимальном режиме продемонстрировала сопоставимый уровень среднеквадратичной ошибки (около 0.0136), что подтверждает способность рекуррентных архитектур учитывать краткосрочные динамические эффекты в поведении ветрового потока. Однако сравнение моделей выявило, что влияние временной компоненты в рассматриваемом датасете не оказывает существенного влияния на итоговый прогноз, что подтверждает корректность решения об исключении временной метки как значимого признака.

Разработанное программное обеспечение обеспечивает полный цикл обработки данных и обучения моделей, включая логирование, визуализацию динамики ошибки и автоматизированную оценку качества на тестовом наборе. Полученные результаты подтверждают применимость нейросетевых методов в задачах прогнозирования энерговыработки и демонстрируют, что относительно простые архитектуры могут обеспечить высокую точность при надлежащей предварительной обработке данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хайкин С. Нейронные сети: полный курс. Пер. с англ. — М.: Вильямс, 2006. — 1104 с.
2. Leskovec J., Rajaraman A., Ullman J. Mining of Massive Datasets. — Cambridge University Press, 2020. — Режим доступа: https://mmds.org — Дата доступа: 02.12.2025.
3. Wind Turbine SCADA Dataset — Режим доступа: https://www.kaggle.com/datasets/berkerisen/wind-turbine-scada-dataset — Дата доступа: 08.12.2025.
4. LSTM Networks: Understanding Long Short-Term Memory Models — Режим доступа: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ — Дата доступа: 10.12.2025.

ПРИЛОЖЕНИЕ А

Wind Turbine Scada Dataset (первые 50 записей)

Date/Time,LV ActivePower (kW),Wind Speed (m/s),Theoretical_Power_Curve (KWh),Wind Direction (°)

01 01 2018 00:00,380.047790527343,5.31133604049682,416.328907824861,259.994903564453

01 01 2018 00:10,453.76919555664,5.67216682434082,519.917511061494,268.64111328125

01 01 2018 00:20,306.376586914062,5.21603679656982,390.900015810951,272.564788818359

01 01 2018 00:30,419.645904541015,5.65967416763305,516.127568975674,271.258087158203

01 01 2018 00:40,380.650695800781,5.57794094085693,491.702971953588,265.674285888671

01 01 2018 00:50,402.391998291015,5.60405206680297,499.436385024805,264.57861328125

01 01 2018 01:00,447.605712890625,5.79300785064697,557.372363290225,266.163604736328

01 01 2018 01:10,387.2421875,5.30604982376098,414.898178826186,257.949493408203

01 01 2018 01:20,463.651214599609,5.58462905883789,493.677652137077,253.480697631835

01 01 2018 01:30,439.725708007812,5.52322816848754,475.706782818068,258.72378540039

01 01 2018 01:40,498.181701660156,5.72411584854125,535.841397042263,251.850997924804

01 01 2018 01:50,526.816223144531,5.93419885635375,603.014076510633,265.504699707031

01 01 2018 02:00,710.587280273437,6.54741382598876,824.662513585882,274.23291015625

01 01 2018
02:10,655.194274902343,6.19974613189697,693.472641075637,266.733184814453

01 01 2018
02:20,754.762512207031,6.50538301467895,808.098138482693,266.76040649414

01 01 2018
02:30,790.173278808593,6.63411617279052,859.459020788565,270.493194580078

01 01 2018
02:40,742.985290527343,6.37891292572021,759.434536596592,266.593292236328

01 01 2018
02:50,748.229614257812,6.4466528892517,785.28100987646,265.571807861328

01 01 2018
03:00,736.647827148437,6.41508293151855,773.172863451736,261.15869140625

01 01 2018
03:10,787.246215820312,6.43753099441528,781.7712157188,257.56021118164

01 01 2018
03:20,722.864074707031,6.22002410888671,700.764699868076,255.926498413085

01 01 2018
03:30,935.033386230468,6.89802598953247,970.736626881787,250.012893676757

01 01 2018
03:40,1220.60900878906,7.60971117019653,1315.04892785216,255.985702514648

01 01 2018
03:50,1053.77197265625,7.28835582733154,1151.26574355584,255.444595336914

01 01 2018
04:00,1493.80798339843,7.94310188293457,1497.58372354361,256.407409667968

01 01 2018
04:10,1724.48803710937,8.37616157531738,1752.19966204818,252.41259765625

01 01 2018
04:20,1636.93505859375,8.23695755004882,1668.47070685152,247.979400634765

01 01 2018
04:30,1385.48803710937,7.87959098815917,1461.81579081391,238.609603881835

01 01 2018
04:40,1098.93200683593,7.10137605667114,1062.28503444311,245.095596313476

01 01 2018
04:50,1021.4580078125,6.95530700683593,995.995854606612,245.410202026367

01 01 2018
05:00,1164.89294433593,7.09829807281494,1060.85971215544,235.227905273437

01 01 2018
05:10,1073.33203125,6.95363092422485,995.250960801046,242.872695922851

01 01 2018
05:20,1165.30798339843,7.24957799911499,1132.4168612641,244.835693359375

01 01 2018
05:30,1177.98999023437,7.29469108581542,1154.36530469206,242.48159790039

01 01 2018
05:40,1170.53601074218,7.37636995315551,1194.8430985043,247.97720336914

01 01 2018
05:50,1145.53601074218,7.44855403900146,1231.43070603717,249.682998657226

01 01 2018
06:00,1114.02697753906,7.2392520904541,1127.43320551345,248.401000976562

01 01 2018
06:10,1153.18505859375,7.32921123504638,1171.35504358957,244.621704101562

01 01 2018
06:20,1125.3310546875,7.13970518112182,1080.13908466205,244.631805419921

01 01 2018
06:30,1228.73205566406,7.47422885894775,1244.63353439737,245.785995483398

01 01 2018
06:40,1021.79302978515,7.03317403793334,1030.99268581181,248.652206420898

01 01 2018
06:50,957.378173828125,6.88645505905151,965.683334443832,244.611694335937

01 01 2018
07:00,909.887817382812,6.88782119750976,966.279104864065,235.84829711914

01 01 2018
07:10,1000.95397949218,7.21643209457397,1116.4718990154,232.842697143554

01 01 2018
07:20,1024.47802734375,7.0685977935791,1047.17023059277,229.933197021484

01 01 2018
07:30,1009.53399658203,6.93829584121704,988.451940715539,230.13670349121

01 01 2018
07:40,899.492980957031,6.53668785095214,820.416658585943,234.933807373046

01 01 2018
07:50,725.110107421875,6.18062496185302,686.636942163399,232.837905883789

01 01 2018
08:00,585.259399414062,5.81682586669921,564.927659543473,240.328796386718

01 01 2018
08:10,443.913909912109,5.45015096664428,454.773587146918,238.12629699707

ПРИЛОЖЕНИЕ В

Классовая структура проекта

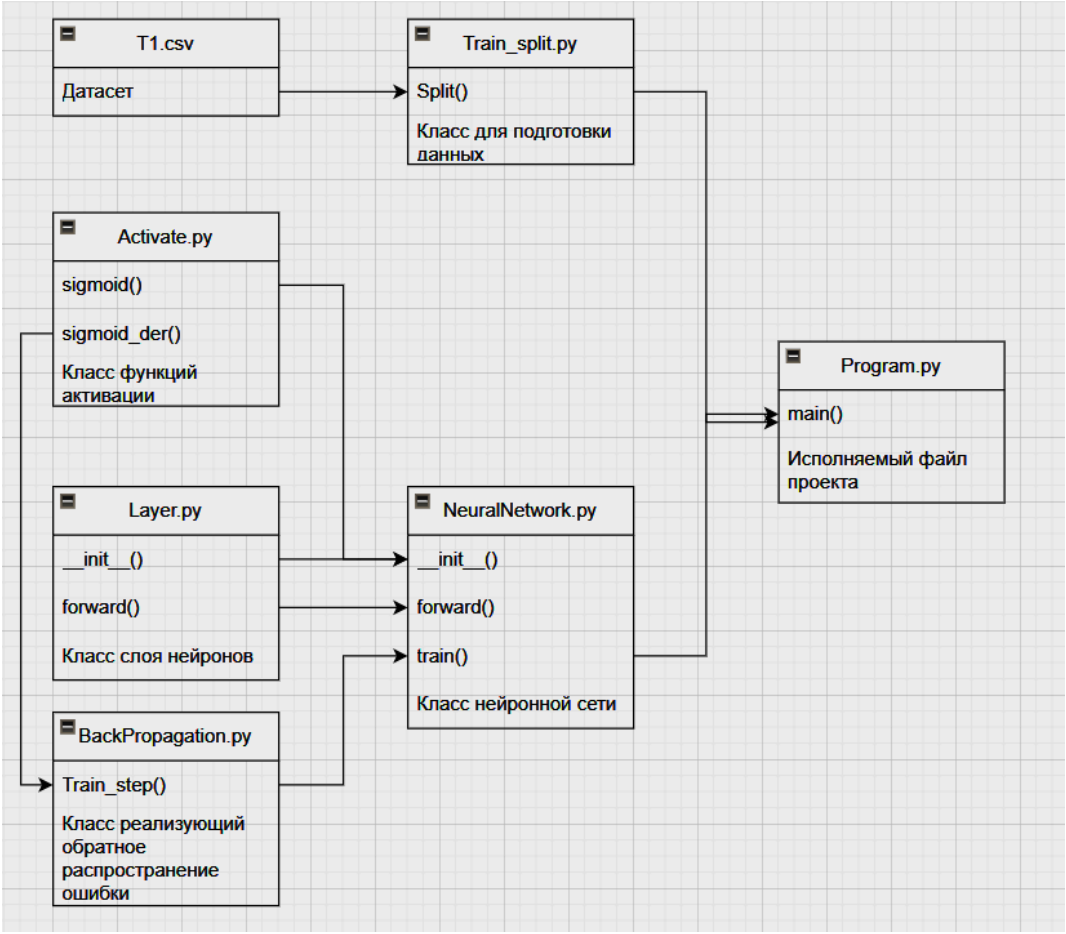

Рисунок В.1 — Структура проекта на стадии разработки.

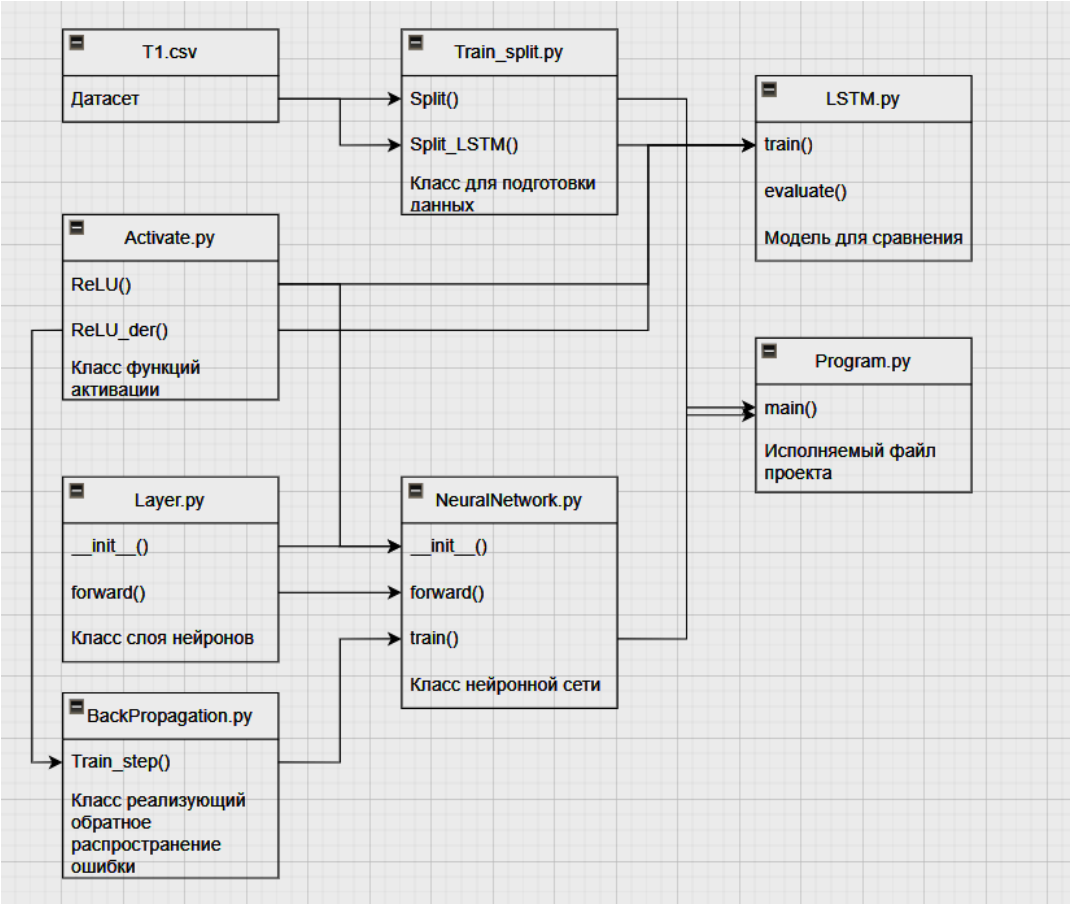

Рисунок В.2 — Финальная структура проекта.

ПРИЛОЖЕНИЕ С

Код основных модулей проекта

NeuralNetwork.py:

```
from Library.Tools.Layer import Layer
from Library.Tools.BackPropagation import train_step


class NeuralNetwork:
    def __init__(self, layers_config, activation_functions):
        self.layers = []
        for i in range(1, len(layers_config)):
            n_inputs = layers_config[i - 1]
            n_neurons = layers_config[i]
            activation = activation_functions[i - 1]
            self.layers.append(Layer(n_inputs, n_neurons, activation))


    def forward(self, X):
        output = X
        for layer in self.layers:
            output = layer.forward(output)
        return output


    def train(self, X, y, epochs=2000, learning_rate=0.1, graph=False):
        for epoch in range(epochs):
            total_error = train_step(self, X, y, learning_rate, graph)
            if graph and (epoch+1) % 100 == 0:
                print(f"Эпоха {epoch+1}/{epochs}, Ошибка: {total_error:.6f}")
```

BackPropagation.py:

```
import numpy as np
from Library.Tools.Activate import reLu_derivative

def train_step(network, X, y, learning_rate, graph):
    outputs = [X]
    for layer in network.layers:
        X = layer.forward(X)
        outputs.append(X)

    error = outputs[-1] - y.values.reshape(-1, 1)
    mse = np.mean(error ** 2)

    for i in reversed(range(len(network.layers))):
        layer = network.layers[i]
        layer_output = outputs[i + 1]
        layer_input = outputs[i]

        if i == len(network.layers) - 1:
            delta = error
        else:
            next_layer = network.layers[i + 1]
            error_back = np.dot(delta, next_layer.weights.T)
            delta = error_back * reLu_derivative(layer_output)

        layer_inputs = np.column_stack([np.ones((layer_input.shape[0], 1)),
layer_input])
```

```
        dw_combined = np.dot(layer_inputs.T, delta) / X.shape[0]
        dw_combined = np.clip(dw_combined, -1.0, 1.0)

        db = dw_combined[0].reshape(1, -1)
        dw = dw_combined[1:]

        layer.weights -= learning_rate * dw
        layer.biases -= learning_rate * db

    return mse
```

Program.py:

```
import numpy as np
import traceback

from Library.Tools.Train_Split import split
from Library.Tools.NeuralNetwork import NeuralNetwork
from Library.Tools.Activate import reLu

try:
    (X_train, X_test), (y_train, y_test), (y_min, y_max) = split(graph=True)

    network = NeuralNetwork(layers_config=[3, 16, 8, 4, 1],
activation_functions=[reLu, reLu, reLu, None])
    network.train(X_train, y_train, graph=True)

    y_pred = network.forward(X_test)
```

```
    test_mse = np.mean((y_pred - y_test) ** 2)
    print(f"Средняя ошибка на тестовом наборе - {test_mse:.6f}")
    input()

except Exception as e:
    print(f"Произошла ошибка: {e}")
    print("\nПодробный traceback:")
    traceback.print_exc()
    input()
```