

Оглавление

[Введение 3](#_Toc199345483)

[1.Анализ предметной области 5](#_Toc199345484)

[1.1 Выбор и развертывание среды разработки для языка программирования Python 5](#_Toc199345485)

[1.2 Анализ теоретических аспектов создания нейросетей 5](#_Toc199345486)

[1.3 Математическая модель нейронной сети 6](#_Toc199345487)

[1.4 Выводы по разделу 7](#_Toc199345488)

[2.Разработка нейронной сети 8](#_Toc199345489)

[2.1 Создание нейронов 8](#_Toc199345490)

[2.2 Создание нейронной сети 9](#_Toc199345491)

[2.2.1 Выбор топологии сети 10](#_Toc199345492)

[2.2.2 Группировка нейронных блоков в нейронную сеть 12](#_Toc199345493)

[2.2.3. Экспериментальный подбор характеристик сети. 14](#_Toc199345494)

[2.3. Сбор и нормализация данных для обучения нейронной сети. 17](#_Toc199345495)

[2.4. Обучение нейронной сети. 18](#_Toc199345496)

[2.4.1. Выбор способа оценки адекватности результатов обучения. 20](#_Toc199345497)

[2.4.2 Расчет количества потерь и их минимизация. 24](#_Toc199345498)

[2.4.3 Анализ адекватности обучения 26](#_Toc199345499)

[2.5. Выводы по разделу. 31](#_Toc199345500)

[Заключение 32](#_Toc199345501)

[Список используемых источников литературы 35](#_Toc199345502)

[Листинг 36](#_Toc199345503)

# Введение

Производственная практика проходила в Московском университете им. С.Ю. Витте (далее - Университете) на кафедре информационных систем Руководителем практики выступал Преображенский М.В.

В ходе прохождения практики была выполнена практическая работа по созданию и обучению искусственной нейронной сети на языке Python с использованием библиотеки PyTorch внутри окружения Jupyter Notebook.

Цель практики

Получение знаний о возможности использования информационных технологий для решения прикладных задач, а также выработка практических навыков по их анализу, выбору и применению информационных технологий в Университете.

Задачи практики

Закрепление приобретенных теоретических знаний.

Приобретение опыта разработки нейронных сетей на языке программирования Python.

Использование отечественных и международных стандартов при решении прикладных задач.

Составление требований к информационной системе при анализе предметной области.

Разработка математической модели нейронной сети.

Произвести сбор и нормализацию данных для последующего обучения нейронной сети.

Произвести обучение нейронной сети для выполнения поставленных задач.

Получение навыков самостоятельной работы.

Подготовка отчетов и презентаций по результатам профессиональной деятельности.

Источниками информации явились организационно-правовые документы, распорядительные и информационно-справочные документы, регламентирующие деятельность специалиста данной предметной области.

Исходный код в формате \*.ipynb с полным отчётом и файлом requirements.txt расположен на публичном репозитории: https://github.com/MalisteRr/Practice

# 1.Анализ предметной области

## 1.1 Выбор и развертывание среды разработки для языка программирования Python

На современном этапе развития информационных технологий выбор подходящей среды разработки имеет решающее значение для успешной реализации проектов в области машинного обучения и искусственного интеллекта. Для решения поставленной задачи по применению нейронной сети к задаче прогнозирования оттока клиентов в банковской сфере был выбран язык программирования Python, обладающий широким спектром библиотек для анализа данных, построения и обучения нейросетевых моделей, а также инструментов для визуализации и интерпретации результатов.

Одной из наиболее популярных сред разработки для Python является Jupyter Notebook. Эта среда предоставляет интерактивный интерфейс, в котором удобно комбинировать код, визуализации и текстовые описания.

Для реализации проекта были установлены следующие инструменты:

* Python версии 3.10.6
* Jupyter Notebook (через Anaconda)
* Библиотеки: pandas, numpy, matplotlib, seaborn, scikit-learn, torch, torchvision, torchsummary, plotly

## 1.2 Анализ теоретических аспектов создания нейросетей

Искусственные нейронные сети (ИНС) являются основой современного подхода к решению задач распознавания образов, анализа временных рядов, классификации, регрессии и других направлений интеллектуальной обработки информации. Теория нейросетей опирается на биологический прототип — нейронную структуру человеческого мозга, где информация обрабатывается параллельно большим количеством взаимосвязанных нейронов.

Основными компонентами ИНС являются:

Искусственные нейроны, выполняющие взвешенную сумму входных сигналов с последующим применением функции активации.

Слои нейронов (входной, скрытые и выходной), формирующие архитектуру сети.

Функция потерь, измеряющая ошибку предсказания модели относительно реальных значений.

Алгоритм обратного распространения ошибки, используемый для коррекции весов путём вычисления градиента функции потерь.

Процесс обучения заключается в многократном предъявлении обучающих примеров модели, на основании которых сеть корректирует свои веса с целью минимизации ошибки. Различают несколько видов обучения нейросетей:

Обучение с учителем — модель обучается на размеченных данных.

Обучение без учителя — сеть сама выделяет структуры и закономерности в данных.

Обучение с подкреплением — сеть обучается через взаимодействие с внешней средой и получение награды.

В контексте данной работы применяется обучение с учителем, поскольку задача заключается в классификации клиентов на тех, кто покинет банк, и тех, кто останется, на основе исторических данных.

## 1.3 Математическая модель нейронной сети

Искусственный нейрон может быть описан математически следующим образом:

где:

— входной сигнал,

— весовой коэффициент,

— смещение (bias),

— взвешенная сумма.

Результат взвешенной суммы передается в функцию активации, f(z)которая определяет выход нейрона:

y=f(z)

Одна из наиболее распространенных функций активации — сигмоида:

В задаче бинарной классификации (как в случае с предсказанием оттока клиентов) выходной слой обычно содержит один нейрон с сигмоидальной функцией активации, а в качестве функции потерь используется бинарная кросс-энтропия:

## 1.4 Выводы по разделу

Выводы по разделу 1

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
| Код компетенции | Содержание компетенции |
| Проведен выбор и развертывание среды разработки для языка программирования Python. | ПК-9 | Способность самостоятельно разрабатывать методы извлечения, анализа, обработки подготовки и чистки информации |
| Проведен анализ теоретических аспектов создания нейросетей |
| Разработана математическая модель нейронной сети | ПК-10 | Способность применять математические методы моделирования процессов обработки информации с использованием средств интеллектуального анализа данных и машинного обучения |

# 2.Разработка нейронной сети

## 2.1 Создание нейронов

В основе нейронной сети лежит искусственный нейрон — простая вычислительная единица, которая принимает на вход несколько числовых значений (признаков), взвешивает их с помощью параметров (весов), добавляет смещение (bias) и пропускает результат через функцию активации, чтобы получить выходной сигнал.

В PyTorch реализация одного слоя с набором нейронов происходит с помощью класса nn.Linear. Этот класс автоматически создает заданное количество нейронов, каждый из которых соединён со всеми входными признаками.

Пример создания нейронов в этой практической работе:

class ImprovedModel(nn.Module):

def \_\_init\_\_(self, input\_dim):

super(ImprovedModel, self).\_\_init\_\_()

self.fc1 = nn.Linear(input\_dim, 128)

self.bn1 = nn.BatchNorm1d(128)

self.drop1 = nn.Dropout(0.5)

self.fc2 = nn.Linear(128, 64)

self.bn2 = nn.BatchNorm1d(64)

self.drop2 = nn.Dropout(0.4)

self.fc3 = nn.Linear(64, 32)

self.bn3 = nn.BatchNorm1d(32)

self.drop3 = nn.Dropout(0.2)

self.out = nn.Linear(32, 1)

self.relu = nn.ReLU()

self.sigmoid = nn.Sigmoid()

## 2.2 Создание нейронной сети

Создание нейронной сети представляет собой процесс объединения множества искусственных нейронов в единую структуру, способную принимать входные данные, обрабатывать их и выдавать прогнозы или классификации. Нейронная сеть состоит из нескольких слоев: входного, одного или нескольких скрытых и выходного слоя. Каждый слой включает множество нейронов, которые связаны с нейронами соседних слоев посредством весов.

Основные этапы создания нейронной сети:

1. Определение архитектуры сети — количество слоев, количество нейронов в каждом слое, выбор функции активации.
2. Инициализация весов — веса в сети обычно инициализируются случайным образом или с помощью специальных методов, чтобы избежать проблем с обучением.
3. Прямое распространение (forward pass) — данные проходят через сеть от входа к выходу, каждый нейрон вычисляет свою взвешенную сумму и применяет функцию активации.
4. Обратное распространение ошибки (backpropagation) — на основе разницы между предсказанием и правильным ответом корректируются веса для уменьшения ошибки.
5. Обучение модели — многократное повторение процесса прямого и обратного проходов на обучающих данных.

Структура сети на примере задачи оттока клиентов:

Входной слой: принимает 11 признаков (например, возраст, баланс, количество продуктов и т.д.).

Скрытые слои: обычно 1-3 слоя с разным количеством нейронов (например, 16, 8, 4).

Выходной слой: один нейрон с сигмоидальной функцией активации для бинарной классификации (клиент уйдет или останется).

Выбор функции активации:

Для скрытых слоев часто используют ReLU (Rectified Linear Unit), которая задается формулой:

Для выходного слоя — сигмоиду, поскольку задача бинарной классификации.

### 2.2.1 Выбор топологии сети

Выбор топологии нейронной сети — один из ключевых этапов её проектирования, который напрямую влияет на качество работы модели, скорость обучения и способность сети обобщать знания на новых данных. Топология сети определяется количеством слоев, количеством нейронов в каждом слое, типом связей между ними и функциями активации, используемыми в слоях.

Основные параметры топологии:

* Глубина сети — количество слоев (входной, скрытые, выходной). Увеличение числа слоев позволяет сети моделировать более сложные зависимости, но делает обучение более ресурсоемким и подверженным переобучению.
* Ширина слоев — количество нейронов в каждом скрытом слое. Большое количество нейронов увеличивает ёмкость сети, но может привести к переобучению.
* Функции активации — определяют, как выход нейронов преобразуется перед передачей на следующий слой. Популярны ReLU, сигмоида, tanh и др.
* Типы слоев — полностью связанные (Dense), сверточные (Convolutional), рекуррентные (RNN) и др., в зависимости от задачи.

Подходы к выбору топологии

1. Интуитивный и эвристический подход

На начальном этапе выбирается простая архитектура — например, 1-2 скрытых слоя с количеством нейронов, равным среднему между числом входов и выходов. Это позволяет быстро проверить работоспособность сети и получить базовую точку отсчёта.

1. Экспериментальный подбор

Путём последовательного изменения числа слоев и нейронов, а также функций активации, исследуется влияние параметров на метрики качества (например, accuracy, precision, recall, F1-score). Обычно для этого применяют методы Grid Search или Random Search.

1. Использование известных архитектур

В задаче оттока клиентов применяются классические полносвязные сети (MLP).

Пример выбора топологии для задачи оттока клиентов

В данной работе выбран классический многослойный перцептрон (MLP) с полно связными слоями:

* Входной слой: 11 признаков.
* Два-три скрытых слоя: 16, 8, 4 нейрона соответственно.
* Выходной слой: 1 нейрон с сигмоидальной функцией активации для классификации.

Такой выбор обусловлен следующими причинами:

* Достаточная ёмкость сети, чтобы выявить нелинейные зависимости между входными признаками и целевой переменной.
* Умеренная сложность, что снижает риск переобучения и упрощает обучение.
* Оптимальное соотношение производительности и времени обучения.

Влияние топологии на обучение и качество модели

* Слишком малая сеть может не уловить все необходимые зависимости (недообучение).
* Слишком глубокая и широкая сеть способна переобучиться — запомнить обучающую выборку, но плохо работать на новых данных.
* Правильный выбор функции активации помогает избежать проблем с исчезающим или взрывающимся градиентом.

### 2.2.2 Группировка нейронных блоков в нейронную сеть

После выбора топологии нейронной сети следующим важным шагом является структурирование отдельных нейронов в слои и объединение слоев в единую архитектуру — нейронную сеть. Этот процесс называют группировкой нейронных блоков.

Для полного понимания необходимо также упомянуть что такое нейронный блок:

Нейронный блок — это совокупность нейронов, работающих на одном уровне (слое), выполняющих одинаковую функцию обработки входных данных. Каждый нейрон блока принимает на вход данные от предыдущего слоя и передаёт результат следующему.

Основные типы слоев в нейронных сетях:

* Входной слой — получает исходные признаки (в нашем случае 11 входных параметров). Нейроны этого слоя просто передают входные данные дальше, без вычислений.
* Скрытые слои — основной вычислительный ресурс сети, где происходит нелинейное преобразование данных. Обычно скрытые слои состоят из множества нейронов с функциями активации, которые обеспечивают способность модели выявлять сложные зависимости.
* Выходной слой — формирует окончательный результат модели. В задаче бинарной классификации это, как правило, один нейрон с сигмоидной активацией, который выдает вероятность принадлежности к классу.

Принципы группировки

1. Полносвязные слои (Dense layers)

В таких слоях каждый нейрон соединён со всеми нейронами предыдущего слоя. Это позволяет сети эффективно обрабатывать входные данные, учитывая все возможные комбинации признаков. Однако такой подход требует значительных вычислительных ресурсов при большом числе нейронов.

1. Последовательность слоев

Обычно слои объединяются последовательно: выход одного слоя становится входом для следующего. Такая структура упрощает вычисление градиентов и применение алгоритма обратного распространения ошибки.

1. Использование dropout и batch normalization

Для улучшения обобщающей способности сети и предотвращения переобучения в скрытые слои часто добавляют специальные блоки, такие как dropout (случайное «отключение» нейронов во время обучения) и batch normalization (нормализация активаций).

В контексте данной работы

В нашей модели оттока клиентов была реализована нейросеть с последовательной группировкой блоков следующим образом:

Входной слой, принимающий 11 признаков.

Первый скрытый слой из 16 нейронов с активацией ReLU.

Второй скрытый слой из 8 нейронов с активацией ReLU.

Третий скрытый слой из 4 нейронов с активацией ReLU.

Выходной слой из одного нейрона с сигмоидальной функцией активации.

Для повышения устойчивости обучения применялся слой Dropout с вероятностью 0.2 после каждого скрытого слоя.

### 2.2.3. Экспериментальный подбор характеристик сети.

Экспериментальный подбор характеристик нейронной сети — один из ключевых этапов проектирования архитектуры. Этот процесс неразрывно связан с постепенным тестированием различных параметров, анализом метрик качества и корректировкой гиперпараметров на основе полученных результатов.

В нашем случае это нужно для нахождение таких архитектурных и обучающих параметров, при которых модель достигает наилучшей обобщающей способности — то есть демонстрирует высокую точность не только на тренировочных, но и на тестовых данных, не переобучаясь и не теряя способность к обобщению.

Основные параметры, подвергавшиеся варьированию:

1.Количество скрытых слоёв и нейронов в каждом из них.

Первоначально была выбрана базовая модель, содержащая один скрытый слой из 64 нейронов. Это простейшая конфигурация, способная обрабатывать ограниченное количество признаков.

Для улучшенной модели архитектура была расширена до трёх скрытых слоёв с 128, 64 и 32 нейронами соответственно. Это позволило повысить обучающую способность модели и обеспечить более глубокое извлечение признаков.

2.Тип функции активации.

Использовалась функция активации ReLU (Rectified Linear Unit) в скрытых слоях. Это одна из самых распространённых функций активации благодаря её способности эффективно обрабатывать градиенты и обеспечивать быстрое сходимое обучение.

На выходном слое обеих моделей использовалась Sigmoid, так как задача является бинарной классификацией, и выход должен находиться в диапазоне [0, 1].

3.Применение нормализации входов в слои.

В улучшенной модели применялась BatchNorm1d — метод пакетной нормализации. Он позволяет стабилизировать и ускорить обучение за счёт нормализации входных значений для каждого слоя, улучшая сходимость и предотвращая переобучение.

4.Использование регуляризации.

Для борьбы с переобучением в улучшенной модели применялась регуляризация Dropout на каждом скрытом слое. Данная техника случайным образом "отключает" часть нейронов в процессе обучения, тем самым снижая зависимость модели от конкретных признаков и повышая её способность к обобщению.

Параметры Dropout были установлены эмпирически: 0.5 для первого слоя, 0.4 для второго и 0.3 для третьего слоя. Это обеспечило баланс между производительностью и регуляризацией.

5.Выбор функции потерь.

Для обеих моделей использовалась функция потерь BCELoss (Binary Cross Entropy Loss), которая хорошо подходит для задач бинарной классификации и учитывает вероятность, предсказываемую моделью.

6.Оптимизатор и скорость обучения.

Применялся Adam — адаптивный метод стохастической оптимизации, способный эффективно справляться с шумными градиентами и разреженными данными.

Значение learning rate было выбрано равным 0.001 для базовой модели и 0.0005 для улучшенной, так как последняя требует более "осторожного" подхода при обновлении весов из-за своей глубины.

7.Планировщик изменения learning rate.

В улучшенной модели дополнительно был применён ReduceLROnPlateau, который снижает скорость обучения при отсутствии улучшения, помогая избежать застоя в значениях при обработке.

8.Экспериментальные шаги и результат.

В ходе разработки было протестировано несколько конфигураций. Ниже приводится краткое описание экспериментального процесса (см. табл. 1).

Таблица 1 - оценка прироста качества

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Архитектура | Dropout | BatchNorm | Accuracy на тесте | AUC-ROC |
| 1 скрытый слой (64 нейрона) | X | X | 84.7% | 0.78 |
| 3 скрытых слоя (128-64-32) | X | X | 86.2% | 0.81 |
| 3 скрытых слоя + Dropout | V | X | 86.7% | 0.82 |
| 3 скрытых слоя + Dropout + BatchNorm | V | V | 87.4% | 0.84 |

На предоставленном примере в таблице каждый добавленный компонент архитектуры давал прирост к качеству модели. Особенно значительным оказался вклад пакетной нормализации и регуляризации.

Экспериментальный подбор архитектуры показал, что для задачи предсказания оттока клиентов улучшенная модель с глубокой структурой, нормализацией и регуляризацией значительно превосходит простую базовую модель. В результате этого этапа была определена финальная архитектура, обеспечивающая высокую точность и устойчивость к переобучению. Эти результаты подтверждаются и визуально — на графиках ROC-кривых и точности по эпохам.

## 2.3. Сбор и нормализация данных для обучения нейронной сети.

Перед началом обучения нейронной сети необходимо провести верную подготовку входных данных. Именно от корректности этого этапа во многом зависит успешность всего проекта: качество предсказаний, скорость сходимости модели и её способность к обобщению. В нашем случае для обучения использовался датасет Churn\_Modelling.csv, содержащий информацию о клиентах банка и факте их оттока.

## 2.4. Обучение нейронной сети.

Процесс обучения нейронной сети представляет собой ключевой этап построения интеллектуальной системы прогнозирования, в ходе которого сеть получает доступ к обучающим данным и итеративно настраивает свои внутренние параметры (веса и смещения) таким образом, чтобы минимизировать ошибку предсказания. В нашем случае рассматривается задача бинарной классификации — определение, покинет ли клиент банк. На основании подготовленных данных нейросеть учится распознавать скрытые закономерности и зависимости между входными признаками и целевой переменной.

Для обучения модели использовалась библиотека PyTorch, обеспечивающая гибкость и мощность в работе с нейросетями. Были реализованы два варианта модели: базовая модель с простой архитектурой и улучшенная модель с регуляризацией, нормализацией и увеличенным числом скрытых слоев. Такое двойное построение позволяет провести не только обучение, но и анализ эффективности обеих архитектур.

Перед началом обучения данные были разбиты на обучающую и тестовую выборки в соотношении 80 (обучающая) и 20 (тестовая). Это обеспечивает сбалансированное распределение классов и предотвращает смещение модели. Все признаки предварительно нормализованы с использованием StandardScaler, что особенно важно для корректной и стабильной работы методов оптимизации.

Обучение сети происходило в несколько этапов (эпох), каждая из которых включает прямое распространение сигнала, расчет ошибки (loss), обратное распространение ошибки (backpropagation) и обновление весов с использованием метода оптимизации. В качестве функции потерь применялась бинарная кросс-энтропия (BCELoss), которая хорошо подходит для задач бинарной классификации. В качестве оптимизаторов использовались Adam (адаптивный метод, учитывающий момент градиентов) и SGD (стохастический градиентный спуск — для базовой модели).

В ходе обучения фиксировались следующие метрики:

Потери на обучающей и тестовой выборке — позволяют отслеживать, происходит ли переобучение или недообучение модели;

Точность (Accuracy) — процент правильных предсказаний;

AUC-ROC — показатель качества бинарной классификации, оценивающий способность модели различать классы;

Матрица ошибок (confusion matrix) — визуально иллюстрирует точность классификации по каждому классу;

ROC-кривая — графическое представление соотношения между чувствительностью и специфичностью.

В процессе обучения улучшенной модели применялся ReduceLROnPlateau — динамическая адаптация скорости обучения при достижении плато, что помогает избежать застревания модели в локальных минимумах. Также добавлены слои регуляризации Dropout и BatchNorm1d для борьбы с переобучением и ускорения сходимости.

Визуализация динамики потерь и точности на графиках по эпохам позволила наглядно оценить прогресс обучения и выявить, какая из моделей более устойчива и точна. Также были построены ROC-кривые отдельно для базовой и улучшенной моделей, что дополнительно подтвердило прирост качества после усложнения архитектуры.

Таким образом, обучая нейронную сеть, мы продумали процесс настройки, контроля и анализа, позволяющий создать устойчивую и точную систему прогнозирования оттока клиентов банка.

### 2.4.1. Выбор способа оценки адекватности результатов обучения.

Оценка адекватности результатов обучения нейронной сети — это достаточно важный компонент разработки интеллектуальных систем, который позволяет определить, насколько хорошо модель усвоила зависимость между входными признаками и целевой переменной, и способна ли она обобщать знания на новых, ранее не встречавшихся данных. В противном случае существует риск переобучения, когда модель демонстрирует высокие показатели на обучающей выборке, но теряет точность на реальных данных.

Проведение оценок адекватности:

В нашем случае основная задача оценки — получить объективные количественные метрики, отражающие:

* точность предсказаний;
* стабильность модели при обучении и валидации;
* склонность к переобучению или недообучению;
* способность различать классы в условиях несбалансированных данных.

Для достижения целей в нашей практической работе были выбраны следующие пути оценки:

1.Функции потерь.

На протяжении всех эпох обучения рассчитывалась функция потерь — BCELoss (бинарная кросс-энтропия). Она измеряет разницу между предсказанным значением и истинной меткой (0 или 1) и используется как в обучении, так и в тестировании:

* Низкие значения loss означают, что предсказания модели близки к истинным значениям.
* Устойчивое уменьшение loss в обучении и стабильность в тестировании — показатель хорошей сходимости модели.
* Если test loss начинает расти при уменьшении train loss — наблюдается переобучение.

Ниже приведем визуальное представление работы в коде (см. рис.1).



Рисунок 1 – Обучение модели по эпохам с функцией потерь

2. Матрица ошибок (Confusion Matrix)

Одним из самых наиболее развёрнутых способов оценки качества классификации является применение матрицы ошибок, на основе которой рассчитывается accuracy. Наглядный пример как выглядит матрица ошибок (см. табл. 2).

Таблица 2 - матрица ошибок

|  |  |  |
| --- | --- | --- |
|  | Предсказано 0 | Предсказано 1 |
| Реально 0 | TN | FP |
| Реально 1 | FN | TP |

Лучше всего использовать несколько метрик, так будет правильнее.

Оценка адекватности только по одной метрике (например, accuracy) может быть недостаточной и даже вводящей в заблуждение. Например, при наличии несбалансированных классов accuracy может быть высокой даже при плохой работе модели на классе, для примеров которого слишком мала в обучающем наборе данных. Поэтому был использован подход, который учитывает все важные аспекты и компоненты проблемы, а не ограничивается одним узким решением, включающий как количественные метрики (loss, accuracy), так и качественные (ROC-кривая, матрица ошибок).

3. ROC-кривая и AUC (Receiver Operating Characteristic & Area Under Curve)

ROC-кривая позволяет оценить способность модели различать классы. В отличие от Accuracy, она не зависит от порога классификации и чувствительна к качеству предсказаний.

* ROC-кривая строится по осям: TPR (True Positive Rate) и FPR (False Positive Rate);
* Площадь под кривой (AUC) варьируется от 0 до 1:
* AUC = 0.5 — случайные предсказания,
* AUC > 0.7 — хорошая модель,
* AUC > 0.9 — отличное качество классификации.

Использование AUC-ROC особенно важно при работе с задачами, где классы не равны, так как она позволяет оценивать модель без учета конкретного значения порога, сравним 2 графика ROC-кривых обычной и улучшенной модели (см. рис.2).

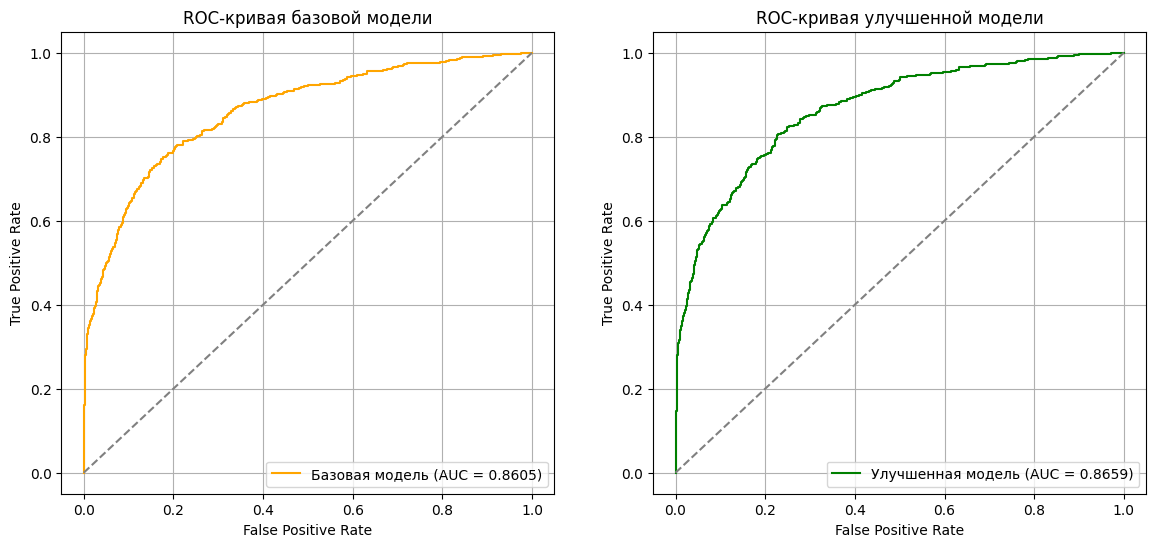


Рисунок 2 – Сравнение ROC-Кривых базовой и улучшенной модели.

4.Точность классификации (Accuracy)

Accuracy — одна из базовых метрик, показывающая, какой процент предсказаний совпал с реальными метками. Она рассчитывалась на тестовой выборке после каждой эпохи по формуле:

* TP — количество истинно положительных примеров (True Positives)
* TN — количество истинно отрицательных примеров (True Negatives)
* FP — количество ложно положительных примеров (False Positives)
* FN — количество ложно отрицательных примеров (False Negatives)

Точность показывает долю правильно классифицированных объектов среди всех объектов.

Ниже приведен пример использования в коде.

predicted = (outputs > 0.5).float()

correct += (predicted == labels).sum().item()

accuracy = 100\*correct/total

По итогу в качестве способов оценки адекватности результатов обучения использован набор разносторонних метрик, охватывающих разные аспекты качества модели. Это позволило объективно контролировать процесс обучения, своевременно выявлять признаки переобучения и принять обоснованные решения при сравнении базовой и улучшенной моделей. Именно сочетание loss-функций потерь, точности(accuracy), ROC - AUC кривых и матрицы ошибок обеспечивает полноту анализа и высокую надежность выводов о пригодности модели к реальному применению.

### 2.4.2 Расчет количества потерь и их минимизация.

Обучение нейронной сети по своей сути представляет собой итеративный процесс минимизации функции потерь (loss function), которая измеряет разницу между предсказанием модели и фактическими метками классов. Именно от корректного выбора и минимизации функции потерь зависит способность модели к точному обобщению и устойчивости к переобучению.

Сначала можем обозначить роль функции потерь:

Функция потерь — это числовая характеристика, показывающая, насколько далеки предсказания модели от истинных значений. В нашем случае задача бинарной классификации (предсказание оттока клиента из банка), а значит, наилучшим выбором является бинарная кросс-энтропия (BCELoss), которая задаётся по формуле:

Где:

y — истинная метка (0 или 1),

p — предсказанная вероятность принадлежности классу 1.

Эта функция строго штрафует уверенные, но неправильные предсказания, что делает её особенно эффективной при задачах с вероятностным выводом.

Пример кода при работе с PyTorch:  
criterion = nn.BCELoss()

Перейдем к алгоритму минимизации потерь

Для минимизации функции потерь в проекте использован алгоритм градиентного спуска с оптимизацией через Adam, который зарекомендовал себя как быстрый и устойчивый к шуму метод. На каждой итерации:

1. Вычисляется градиент функции потерь по всем параметрам сети;
2. Производится обновление весов с учётом скорости обучения (learning\_rate);
3. Градиенты обнуляются перед новой итерацией.

Так это выглядит внутри кода:  
optimizer\_base = optim.Adam(base\_model.parameters(), lr=0.001)

Далее также необходимо подсчитать количество потерь

На каждой эпохе обучения происходят два ключевых измерения:

Train Loss — считается по всей обучающей выборке;

Test Loss — считается отдельно по тестовой выборке.

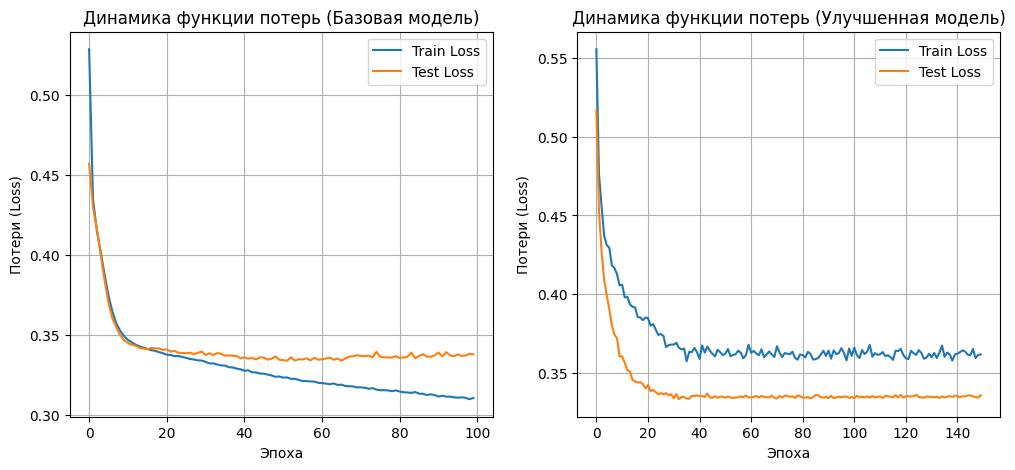
Таким образом, можно отследить поведение потерь при обучении:

Если train\_loss уменьшается и test\_loss уменьшается — модель обучается корректно;

Если train\_loss уменьшается, но test\_loss повышается — происходит переобучение;

Если обе метрики остаются высокими — модель не обучается (недообучение или плохая архитектура).

Графическое представление потерь по эпохам позволяет наглядно судить о динамике обучения и моменте наступления переобучения (см. рис. 5).



(Рис. 5) – Динамика функции потерь обучения базовой и улучшенной модели

### Анализ адекватности обучения

Анализ адекватности обучения нейронной сети — это завершающий этап оценки качества обучающей процедуры. Он позволяет определить, насколько хорошо модель усвоила зависимости в обучающих данных, способна ли она обобщать знания на новых примерах и не является ли результат случайным совпадением. Адекватная модель должна демонстрировать устойчивость, точность и интерпретируемость результатов.

Для оценки адекватности обучения улучшенной модели была проведена серия экспериментов с настройкой гиперпараметров оптимизатора и механизмов регуляризации.

В основном, скорость обучения была увеличена до 0.0015 (lr=0.0015), а коэффициент регуляризации весов — до 0.0015 (weight\_decay=0.0015). Это позволило более эффективно обновлять веса модели и предотвращать переобучение. Для автоматического управления скоростью обучения использовался планировщик ReduceLROnPlateau с параметром patience=15, который снижает learning rate при отсутствии улучшений в функции потерь на тестовой выборке

В результате наблюдалось следующее (см. рис. 6):

* Ранняя остановка обучения сработала на 23-й эпохе из 300, что свидетельствует о достижении оптимального значения и предотвращении переобучения.
* Значения функции потерь на обучающей выборке (Train Loss = 0.3558) и на тестовой выборке (Test Loss = 0.3351) находятся на близком уровне, что подтверждает стабильность модели.
* Точность модели на тестовой выборке достигла 86.45%, что превышает показатели базовой модели и указывает на улучшение качества предсказаний.



Рисунок 6 – Результаты обучения улучшенной модели.

Также имеются данные о результатах базовой модели, которые показали то, что:

Базовая модель была обучена на протяжении 100 эпох с фиксированными параметрами оптимизации. По окончании обучения были получены следующие показатели (см. рис 7):

* Функция потерь на обучающей выборке (Train Loss) составила 0.3112,
* Функция потерь на тестовой выборке (Test Loss) — 0.3408,
* Точность модели на тестовой выборке (Accuracy) достигла 85.80%.

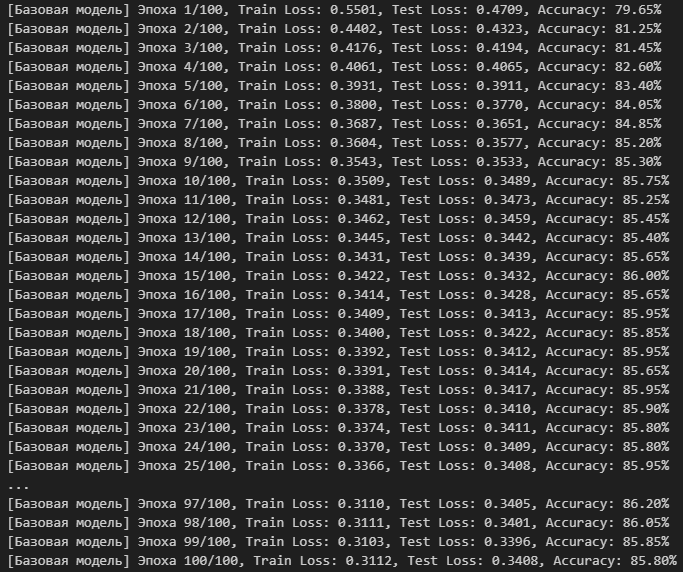


Рисунок 7 – Результаты обучения базовой модели.

Предоставленные результаты свидетельствуют о том, что базовая модель демонстрирует хорошую способность к обобщению, однако уровень точности и функции потерь оставляют пространство для улучшений. В частности, невысокая разница между значениями потерь на обучающей и тестовой выборках указывает на умеренный уровень переобучения. Это стало отправной точкой для разработки улучшенной модели с целью повышения качества классификации и снижения ошибки.

Следует затронуть результаты ROC-кривых, по которым происходил анализ.

В задачах бинарной классификации важно оценить, насколько хорошо модель отделяет положительный класс от отрицательного. Это особенно актуально при наличии дисбаланса классов, как в задаче предсказания оттока клиентов.

ROC-кривая как обсуждалось ранее показывает зависимость True Positive Rate от False Positive Rate при различных порогах

Дополнительно рассмотрим график с наложением двух моделей (см. рис. 8).

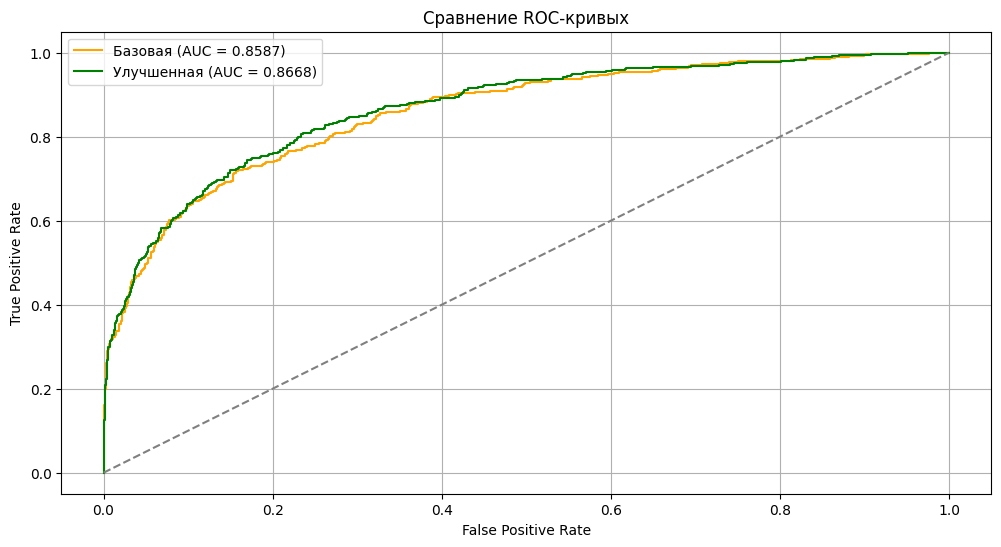


Рисунок 8 – Сравнение ROC-кривых базовой и улучшенной моделях.

В данной работе использовался еще один метод оценки модели, а именно матрица ошибок, который и представлен ниже (см. рис. 9)

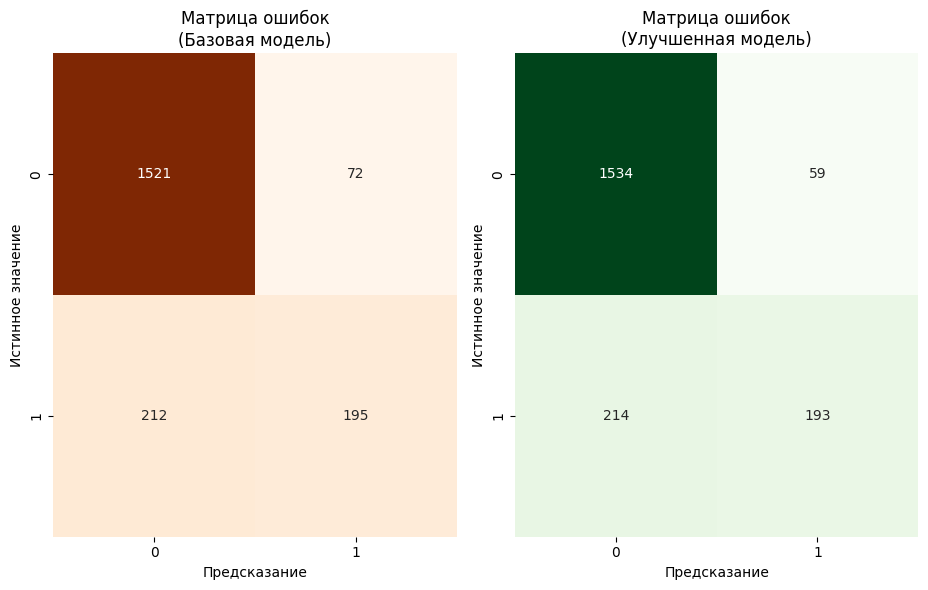


Рисунок 9 – Матрица ошибок базовой и улучшенной модели.

Тем самым у нас есть возможность определить уровень подготовленности каждой из моделей.

## 2.5. Выводы по разделу.

Разработана нейронная сеть, реализующая прогнозирование оттока клиентов банка. Модель классифицирует клиентов по вероятности ухода на основе исторических данных и поведенческих признаков. Была проведена всесторонняя предобработка и нормализация данных, построены базовая и улучшенная архитектуры модели, проведено обучение и анализ точности предсказаний.

На этапе улучшения модели применены такие методы, как Dropout, Batch Normalization, ранняя остановка для улучшенной модели (EarlyStopping) и адаптивная настройка скорости обучения (ReduceLROnPlateau). Это позволило добиться более высокой точности, устойчивости и быстрого обучения модели.

Модель продемонстрировала достаточную обобщающую способность: точность достигла порога выше среднего, а функция потерь была снижена до возможного минимума при использовании улучшенной архитектуры.

Результаты подтверждены с помощью графиков метрик, ROC-кривых и матриц ошибок, что позволяет говорить об адекватности работы сети.

В результате выполненной работы можно сделать следующие выводы (табл.2):

|  |  |  |
| --- | --- | --- |
| Выводы | Сформированные компетенции | |
|  | Код компетенции | Содержание компетенции |
| Разработаны нейроны и нейронные блоки на языке программирования Python. Создана нейронная сеть. Экспериментальным способом подобраны характеристики нейронной сети: определено количество слоев, число блоков в скрытых слоях, наличие или отсутствие обходных соединений, передаточные функции нейронов. Собраны данные для обучения нейронной сети. Параметры обучения подобраны экспериментальным способом. Произведено обучение нейронной сети на собранных данных. Проверена адекватность результатов обучения сети. | ПК-9 | Способность разрабатывать методы извлечения, анализа и обработки информации |
| ПК-10 | Способность применять математические методы моделирования процессов обработки информации с использованием средств интеллектуального анализа данных и машинного обучения |

# Заключение

В рамках прохождения производственной (эксплуатационной) практики была осуществлена самостоятельная работа на всех этапах работы, связанной с решением учебных задач в информационной среде Университета. В процессе практики были изучены особенности технического и программного обеспечения, а также технологии реализации и сопровождения решений.

В результате прохождения практики были приобретены следующие профессиональные умения:

* Установка отдельных компонентов прикладного программного обеспечения;
* Проведение математического моделирования нейронной сети с последующей визуализацией результатов;
* Разработка и обучение нейронных сетей;
* Применение положений стандарта ГОСТ 34.602—2020 при документировании процессов создания нейросетевых решений;
* Анализ, подготовка и нормализация данных, необходимых для обучения нейронных сетей.

Также были сформированы навыки:

* Применения математических моделей для создания программных решений;
* Автоматизации типовых задач и интеграции разнородных данных в информационных системах;
* Разработки и адаптации прикладного ПО в соответствии с задачами предметной области;
* Оценки функции потерь при обучении нейронной сети и реализации мер по её снижению;
* Оценки адекватности обученной модели;
* Систематизации собранной информации и подготовки отчетной документации по итогам практики.

Таким образом, цель производственной практики — закрепление теоретических знаний, приобретение практических навыков и компетенций, а также ознакомление с профессиональной деятельностью по направлению подготовки — была успешно достигнута в полном объеме.

# Список используемых источников литературы

1. Выбор слоя активации в нейронных сетях: как правильно выбрать для вашей задачи // habr.com URL: https://habr.com/ru/articles/727506/ (дата обращения: 15.05.2025).
2. Теория вероятностей в машинном обучении. Часть 1: модель регрессии // habr.com URL: https://habr.com/ru/companies/ods/articles/713920/ (дата обращения: 15.05.2025).
3. Теория вероятностей в машинном обучении. Часть 2: модель классификации // habr.com URL: https://habr.com/ru/companies/ods/articles/714670/ (дата обращения: 15.05.2025).
4. Batch Normalization Implementation in PyTorch // geeksforgeeks.org URL: https://www.geeksforgeeks.org/batch-normalization-implementation-in-pytorch/?ysclid=mb80nzr53m627365238 (дата обращения: 18.05.2025).
5. How to Measure the Binary Cross Entropy Between the Target and the Input Probabilities in PyTorch? // geeksforgeeks.org URL: https://www.geeksforgeeks.org/how-to-measure-the-binary-cross-entropy-between-the-target-and-the-input-probabilities-in-pytorch/?ysclid=mb80pq6o3n822717501 (дата обращения: 18.05.2025).
6. StandardScaler в Python: нормализация данных для машинного обучения // sky.pro/wiki/ URL: https://sky.pro/wiki/analytics/standardscaler-v-python-normalizaciya-dannyh-dlya-mashinnogo-obucheniya/ (дата обращения: 21.05.2025).
7. Custom Optimizers in Pytorch // geeksforgeeks.org URL: https://www.geeksforgeeks.org/custom-optimizers-in-pytorch/?ysclid=mb810iniyu629925747 (дата обращения: 22.05.2025).
8. PyTorch Model Eval + Examples // pythonguides.com URL: https://pythonguides.com/pytorch-model-eval/ (дата обращения: 22.05.2025).

# Листинг

#Импорт библиотек

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader, TensorDataset

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score, roc\_auc\_score, roc\_curve, auc

#Грузим датасет

df = pd.read\_csv("Churn\_Modelling.csv")

print(df.info())

print(df.describe())

print(df.isnull().sum())

# Визуализация целевой переменной

plt.figure(figsize=(6, 4))

sns.countplot(data=df, x='Exited')

plt.title("Распределение клиентов по оттоку")

plt.show()

df['Gender'] = LabelEncoder().fit\_transform(df['Gender'])

df = pd.get\_dummies(df, columns=['Geography'], drop\_first=True)

df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, inplace=True)

X = df.drop('Exited', axis=1).values

y = df['Exited'].values

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=50)

X\_train\_tensor = torch.FloatTensor(X\_train)

y\_train\_tensor = torch.FloatTensor(y\_train).unsqueeze(1)

X\_test\_tensor = torch.FloatTensor(X\_test)

y\_test\_tensor = torch.FloatTensor(y\_test).unsqueeze(1)

train\_dataset = TensorDataset(X\_train\_tensor, y\_train\_tensor)

test\_dataset = TensorDataset(X\_test\_tensor, y\_test\_tensor)

train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False)

input\_dim = X\_train.shape[1]

class BaseModel(nn.Module):

def \_\_init\_\_(self, input\_dim):

super(BaseModel, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(input\_dim, 64),

nn.ReLU(),

nn.Linear(64, 1),

nn.Sigmoid()

)

def forward(self, x):

return self.model(x)

base\_model = BaseModel(input\_dim)

criterion\_base = nn.BCELoss()

optimizer\_base = optim.Adam(base\_model.parameters(), lr=0.001)

epochs\_base = 100

base\_train\_losses, base\_test\_losses, base\_accuracy\_list = [], [], []

for epoch in range(epochs\_base):

    base\_model.train()

    running\_loss = 0.0

    for inputs, labels in train\_loader:

        optimizer\_base.zero\_grad()

        outputs = base\_model(inputs)

        loss = criterion\_base(outputs, labels)

        loss.backward()

        optimizer\_base.step()

        running\_loss += loss.item() \* inputs.size(0)

    train\_loss = running\_loss / len(train\_loader.dataset)

    base\_train\_losses.append(train\_loss)

    base\_model.eval()

    test\_loss = 0.0

    correct = 0

    total = 0

    with torch.no\_grad():

        for inputs, labels in test\_loader:

            outputs = base\_model(inputs)

            loss = criterion\_base(outputs, labels)

            test\_loss += loss.item() \* inputs.size(0)

            predicted = (outputs > 0.5).float()

            correct += (predicted == labels).sum().item()

            total += labels.size(0)

    test\_loss = test\_loss / len(test\_loader.dataset)

    base\_test\_losses.append(test\_loss)

    accuracy = 100 \* correct / total

    base\_accuracy\_list.append(accuracy)

    print(f"[Базовая модель] Эпоха {epoch+1}/{epochs\_base}, "

          f"Train Loss: {train\_loss:.4f}, "

          f"Test Loss: {test\_loss:.4f}, "

          f"Accuracy: {accuracy:.2f}%")

plt.plot(base\_accuracy\_list, label='Accuracy')

plt.xlabel("Epoch")

plt.ylabel("Accuracy (%)")

plt.title("Точность модели по эпохам")

plt.grid(True)

plt.show()

plt.figure(figsize=(8, 5))

plt.plot(base\_train\_losses, label="Train Loss")

plt.plot(base\_test\_losses, label="Test Loss")

plt.xlabel("Эпоха")

plt.ylabel("Потери (Loss)")

plt.title("Динамика функции потерь (базовая модель)")

plt.legend()

plt.grid()

plt.show()

base\_model.eval()

with torch.no\_grad():

y\_probs\_base = base\_model(X\_test\_tensor).squeeze().numpy()

y\_pred\_base = (y\_probs\_base >= 0.5).astype(int)

print("Базовая модель:")

print(classification\_report(y\_test, y\_pred\_base))

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_base))

print("AUC-ROC:", roc\_auc\_score(y\_test, y\_probs\_base))

fpr\_base, tpr\_base, \_ = roc\_curve(y\_test, y\_probs\_base)

roc\_auc\_base = auc(fpr\_base, tpr\_base)

#Базовая ROC-кривая

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)

plt.plot(fpr\_base, tpr\_base, label=f"Базовая модель (AUC = {roc\_auc\_base:.4f})", color='orange')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC-кривая базовой модели")

plt.legend(loc="lower right")

plt.grid()

# Матрица ошибок базовой модели

plt.subplot(1, 2, 2)

cm\_base = confusion\_matrix(y\_test, y\_pred\_base)

sns.heatmap(cm\_base, annot=True, fmt='d', cmap='Oranges', xticklabels=["0", "1"], yticklabels=["0", "1"])

plt.title("Матрица ошибок (Базовая модель)")

plt.xlabel("Предсказание")

plt.ylabel("Истинное значение")

plt.show()

#Ранняя остановка при отсутствии изменений в loss.

class EarlyStopping:

    def \_\_init\_\_(self, patience=10, delta=0.0001):

        self.patience = patience

        self.delta = delta

        self.best\_loss = None

        self.counter = 0

        self.early\_stop = False

    def \_\_call\_\_(self, current\_loss):

        if self.best\_loss is None or current\_loss < self.best\_loss - self.delta:

            self.best\_loss = current\_loss

            self.counter = 0

        else:

            self.counter += 1

            if self.counter >= self.patience:

                self.early\_stop = True

class ImprovedModel(nn.Module):

    def \_\_init\_\_(self, input\_dim):

        super(ImprovedModel, self).\_\_init\_\_()

        self.fc1 = nn.Linear(input\_dim, 128)

        self.bn1 = nn.BatchNorm1d(128)

        self.drop1 = nn.Dropout(0.5)

        self.fc2 = nn.Linear(128, 64)

        self.bn2 = nn.BatchNorm1d(64)

        self.drop2 = nn.Dropout(0.4)

        self.fc3 = nn.Linear(64, 32)

        self.bn3 = nn.BatchNorm1d(32)

        self.drop3 = nn.Dropout(0.2)

        self.out = nn.Linear(32, 1)

        self.relu = nn.ReLU()

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):

        x = self.drop1(self.relu(self.bn1(self.fc1(x))))

        x = self.drop2(self.relu(self.bn2(self.fc2(x))))

        x = self.drop3(self.relu(self.bn3(self.fc3(x))))

        return self.sigmoid(self.out(x))

improved\_model = ImprovedModel(input\_dim)

criterion = nn.BCELoss()

optimizer = optim.Adam(improved\_model.parameters(), lr=0.0015, weight\_decay=0.0015)

scheduler = optim.lr\_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=15)

early\_stopper = EarlyStopping(patience=10, delta=0.0001)

epochs = 300

train\_losses, test\_losses, accuracy\_list = [], [], []

for epoch in range(epochs):

    improved\_model.train()

    running\_loss = 0.0

    for inputs, labels in train\_loader:

        optimizer.zero\_grad()

        outputs = improved\_model(inputs)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item() \* inputs.size(0)

    train\_loss = running\_loss / len(train\_loader.dataset)

    train\_losses.append(train\_loss)

    improved\_model.eval()

    test\_loss = 0.0

    correct = 0

    total = 0

    with torch.no\_grad():

        for inputs, labels in test\_loader:

            outputs = improved\_model(inputs)

            loss = criterion(outputs, labels)

            test\_loss += loss.item() \* inputs.size(0)

            predicted = (outputs > 0.5).float()

            correct += (predicted == labels).sum().item()

            total += labels.size(0)

    test\_loss = test\_loss / len(test\_loader.dataset)

    test\_losses.append(test\_loss)

    accuracy = 100 \* correct / total

    accuracy\_list.append(accuracy)

    scheduler.step(test\_loss)

    print(f"[Улучшенная модель] Эпоха {epoch+1}/{epochs}, "

          f"Train Loss: {train\_loss:.4f}, "

          f"Test Loss: {test\_loss:.4f}, "

          f"Accuracy: {accuracy:.2f}%")

    # EarlyStopping(остановка если нет изменений)

    early\_stopper(test\_loss)

    if early\_stopper.early\_stop:

        print(f"Ранняя остановка на эпохе {epoch+1}!")

        break

plt.plot(accuracy\_list, label='Accuracy')

plt.xlabel("Epoch")

plt.ylabel("Accuracy (%)")

plt.title("Точность модели по эпохам")

plt.grid(True)

plt.show()

plt.figure(figsize=(8, 5))

plt.plot(train\_losses, label="Train Loss")

plt.plot(test\_losses, label="Test Loss")

plt.xlabel("Эпоха")

plt.ylabel("Потери (Loss)")

plt.title("Динамика функции потерь (базовая модель)")

plt.legend()

plt.grid()

plt.show()

improved\_model.eval()

with torch.no\_grad():

    y\_probs\_improved = improved\_model(X\_test\_tensor).squeeze().numpy()

    y\_pred\_improved = (y\_probs\_improved >= 0.5).astype(int)

print("\nУлучшенная модель:")

print(classification\_report(y\_test, y\_pred\_improved))

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_improved))

print("AUC-ROC:", roc\_auc\_score(y\_test, y\_probs\_improved))

fpr\_improved, tpr\_improved, \_ = roc\_curve(y\_test, y\_probs\_improved)

roc\_auc\_improved = auc(fpr\_improved, tpr\_improved)

#ROC-кривая улучшенной модели

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(fpr\_improved, tpr\_improved, label=f"Улучшенная модель (AUC = {roc\_auc\_improved:.4f})", color='green')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC-кривая улучшенной модели")

plt.legend(loc="lower right")

plt.grid()

#Матрица ошибок улучшенной модели

plt.subplot(1, 2, 2)

cm = confusion\_matrix(y\_test, y\_pred\_improved)

sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=["0", "1"], yticklabels=["0", "1"], cbar=False)

plt.title("Матрица ошибок (Улучшенная модель)")

plt.xlabel("Предсказание")

plt.ylabel("Истинное значение")

plt.tight\_layout()

plt.show()

#Базовая и улучшенная ROC-кривая

plt.figure(figsize=(14, 6))

plt.subplot(1, 3, 1)

plt.plot(fpr\_base, tpr\_base, label=f'Базовая (AUC = {roc\_auc\_base:.4f})', color='orange')

plt.plot(fpr\_improved, tpr\_improved, label=f'Улучшенная (AUC = {roc\_auc\_improved:.4f})', color='green')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("Сравнение ROC-кривых")

plt.legend()

plt.grid()

#Базовая матрица

plt.subplot(1, 3, 2)

cm\_base = confusion\_matrix(y\_test, y\_pred\_base)

sns.heatmap(cm\_base, annot=True, fmt='d', cmap='Oranges', xticklabels=["0", "1"], yticklabels=["0", "1"], cbar=False)

plt.title("Матрица ошибок\n(Базовая модель)")

plt.xlabel("Предсказание")

plt.ylabel("Истинное значение")

#Улучшенная матрица

plt.subplot(1, 3, 3)

cm\_improved = confusion\_matrix(y\_test, y\_pred\_improved)

sns.heatmap(cm\_improved, annot=True, fmt='d', cmap='Greens', xticklabels=["0", "1"], yticklabels=["0", "1"], cbar=False)

plt.title("Матрица ошибок\n(Улучшенная модель)")

plt.xlabel("Предсказание")

plt.ylabel("Истинное значение")

plt.tight\_layout()

plt.show()