

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ
по дисциплине «Нейросетевое моделирование технологических процессов»
для студентов направления подготовки
04.03.01 Химия

Ставрополь, 2026

СОДЕРЖАНИЕ

Оглавление

ВВЕДЕНИЕ	3
Лабораторная работа №1	6
Лабораторная работа №2	15
Лабораторная работа №3	24
Лабораторная работа №4	34
Лабораторная работа №5	45
Лабораторная работа №6	58
Лабораторная работа №7	72
Лабораторная работа №8	85
Лабораторная работа №9	99

ВВЕДЕНИЕ

Целью освоения дисциплины «Нейросетевое моделирование технологических процессов» является формирование представлений и фундаментальных знаний в области применения искусственных нейронных сетей для анализа, математического описания, оптимизации и предиктивного управления химико-технологическими процессами.

Задачами освоения дисциплины «Нейросетевое моделирование технологических процессов» являются:

- освоение теоретических основ архитектур нейронных сетей и алгоритмов их обучения;
- развитие навыков сбора, предобработки и инженерной трансформации данных химических производств и лабораторных исследований;
- обучение построению, валидации и интерпретации нейросетевых моделей для задач кинетики, тепломассопереноса, контроля качества и оптимизации режимов;
- формирование компетенций по интеграции обученных моделей в производственную информационную среду и автоматизированные системы управления технологическим процессом.

Дисциплина «Нейросетевое моделирование технологических процессов» Б1.О.05.03 является дисциплиной первого блока, ее освоение происходит в 3 семестре. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесенных с планируемыми результатами освоения образовательной программы:

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-1. Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач.	ИД-1 ук-1 Обладает базовыми знаниями в области системного анализа и методов поиска научно-технической информации.	Знать: принципы системного подхода к описанию химико-технологических процессов, классификацию источников экспериментальных и технологических данных, критерии достоверности и релевантности научной информации.
	ИД-2 ук-1 Использует системный подход для анализа научно-технической информации и постановки задач моделирования.	Уметь: выполнять целенаправленный поиск литературы и открытых датасетов, структурировать разнородные данные, формулировать задачу нейросетевого моделирования в контексте конкретных технологических или исследовательских проблем.
	ИД-3 ук-1 Владеет навыками критического анализа источников и синтеза данных для выбора оптимальных подходов к моделированию технологических процессов.	Владеть: методами сравнительного анализа архитектур нейронных сетей, навыками синтеза теоретических положений химии, математики и data-science для обоснования выбранной вычислительной стратегии.
ОПК-3. Способен	ИД-1 омп-3 Демонстрирует	Знать: принципы построения,

применять и модифицировать математические модели для решения задач в области профессиональной деятельности.	навыки применения математического аппарата для построения адекватных математических моделей реальных процессов, объектов и систем для решения задач в области своей профессиональной деятельности.	обучения, валидации и интерпретации искусственных нейронных сетей как суррогатных математических моделей химико-технологических систем и лабораторных установок.
	ИД-2 опк-3 Использует и адаптирует существующие математические модели для решения прикладных задач.	Уметь: модифицировать архитектуры нейронных сетей, настраивать гиперпараметры, применять методы регуляризации и трансферного обучения для адаптации готовых алгоритмических решений под специфику экспериментальных химических данных.
	ИД-3 опк-3 Применяет на практике математические модели и компьютерные технологии для решения различных задач в области своей профессиональной деятельности.	Владеть: практическими навыками программной реализации, отладки и тестирования нейросетевых моделей в экосистеме Python для прогнозирования параметров процессов, оптимизации режимов и анализа спектрометрических данных.
ОПК-5. Способен проводить физико-химический анализ веществ и материалов, интерпретировать результаты экспериментальных исследований.	ИД-1 опк-5 Обладает знаниями методов физико-химического анализа и принципов обработки экспериментальных данных.	Знать: природу спектроскопических, хроматографических и термических сигналов, источники инструментальных погрешностей, методы цифровой фильтрации и нормализации аналитических данных.
	ИД-2 опк-5 Использует нейросетевые алгоритмы для анализа, классификации и интерпретации спектральных, хроматографических и микроскопических данных.	Уметь: применять сверточные нейронные сети и методы снижения размерности для автоматической идентификации веществ, выделения значимых спектральных признаков и количественной оценки концентраций.
	ИД-3 опк-5 Владеет навыками интеграции инструментальных данных в вычислительные конвейеры для автоматизации анализа веществ и материалов.	Владеть: инструментами построения воспроизводимых пайплайнов обработки сырых сигналов, методами валидации моделей на независимых выборках, навыками интерпретации результатов в терминах физической химии.
ОПК-6. Способен разрабатывать и оптимизировать технологические процессы и схемы производств,	ИД-1 опк-6 Обладает знаниями основ проектирования и оптимизации химико-технологических процессов, а также методов оценки их эффективности.	Знать: принципы технологического проектирования, критерии эффективности и безопасности процессов, ограничения детерминированных моделей при описании нестационарных и

оценивать их эффективность и безопасность.		многофакторных систем.
	ИД-2 опк-6 Использует суррогатные нейросетевые модели для прогнозирования параметров процессов и поиска оптимальных режимов.	Уметь: строить модели на основе данных автоматизированных систем управления технологическим процессом или лабораторных экспериментов, проводить многокритериальную оптимизацию, оценивать влияние входных параметров на целевые показатели процесса.
	ИД-3 опк-6 Владеет навыками применения методов машинного обучения для многокритериальной оптимизации технологических схем и предиктивного управления безопасностью.	Владеть: методами интерпретации решений моделей, подходами к квантификации неопределенности прогнозов, навыками формирования рекомендаций по корректировке режимов в реальном времени.

Целью лабораторных работ является формирование профессиональной деятельности будущего химика-технолога через практическое освоение современных инструментов искусственного интеллекта. В ходе выполнения практикума студент приобретает компетенции, необходимые для применения нейросетевых методов при анализе, моделировании и оптимизации химико-технологических процессов.

Основная цель, которая ставится перед студентом при выполнении задания – практическое освоение всех этапов работы с цифровым инструментарием (программным пакетом прикладных программ), а именно:

1. **Подготовка вычислительной среды:** настройка доступа к облачным ресурсам, импорт необходимых библиотек, проверка корректности выполнения базовых операций.
2. **Работа с данными:** генерация или загрузка технологических параметров, предобработка (масштабирование, кодирование), визуальный разведочный анализ.
3. **Построение модели:** выбор архитектуры нейронной сети, конфигурация слоев и функций активации, компиляция модели.
4. **Обучение и валидация:** запуск процесса обучения, подбор гиперпараметров (оптимизатор, скорость обучения, размер батча), контроль сходимости.
5. **Анализ результатов:** расчет метрик качества, интерпретация прогнозов в контексте физико-химической сущности процесса, формулировка технологических выводов.

Лабораторные работы выполняются индивидуально в соответствии с вариантом задания. Варианты различаются параметрами моделируемого процесса (диапазоны температур, давлений, концентраций), видом целевой функции (выход продукта, селективность, конверсия) или архитектурными ограничениями нейронной сети.

Требования к отчету

Выполнив лабораторную работу, студент оформляет отчет, который состоит из следующих разделов:

1. Тема и цель работы.

- Краткая формулировка изучаемого метода или инструмента (например, «Многослойный перцептрон для регрессии», «Влияние функции активации на сходимость обучения»).
- Цель работы в контексте профессиональной задачи (например, «Освоение методики прогнозирования выхода продукта с использованием нейросетевого моделирования»).

2. Условия задания.

- Номер варианта и его текстовое описание.
- Исходные параметры технологического процесса (математическая модель или диапазоны входных величин).
- Конфигурация нейронной сети (количество слоев, нейронов, тип активации) или исследуемые гиперпараметры.

3. Разработанные документы.

- Скриншоты ключевых этапов выполнения работы в среде Google Colab (фрагменты кода с комментариями, графики обучения, визуализация предсказаний).
- Таблицы с результатами расчетов (значения метрик качества: MSE, MAE, R^2).
- Краткий комментарий к полученным результатам (анализ графиков, объяснение причин расхождений, оценка адекватности модели).

4. Выводы.

- Обобщение освоенных практических навыков работы с инструментом.
- Оценка применимости рассмотренного метода для задач химической технологии.
- Перспективы использования полученных результатов в профессиональной деятельности (например, для оптимизации режимов реактора или ускорения скрининга катализаторов).

Порядок защиты лабораторных работ

При защите лабораторной работы студент должен не только продемонстрировать корректность выполнения кода, но и уметь пояснять логику своей работы. Необходимо отвечать на вопросы и аргументированно обосновывать принятые решения.

СТРУКТУРА И СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1

«Перцептрон для прогнозирования выхода химической реакции» – базовое нейросетевое моделирование регрессионной зависимости

Цель: Создать программу для автоматического обучения простейшей нейронной сети (перцептрона), прогнозирующей выход продукта химической реакции на основе температуры и давления.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 1: Перцептрон для химической реакции

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст» (чтобы это была текстовая ячейка), и скопируйте этот текст:

Лабораторная работа №1

Перцептрон для прогнозирования выхода химической реакции

****Студент:**** [Ваша фамилия и имя]

****Дата:**** [Сегодняшняя дата]

****Вариант:**** [Номер варианта]

****Задание:****

Создать программу, которая:

1. Генерирует синтетические данные химического процесса (температура, давление → выход продукта)
2. Строит простейшую нейронную сеть (перцептрон)
3. Обучает сеть на данных
4. Визуализирует результаты прогнозирования
5. Оценивает точность модели по метрике MSE

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» (нажмите + Код) и скопируйте:

Импортируем нужные библиотеки

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.metrics import mean_squared_error, r2_score

print(" Библиотеки загружены!")

print(f"Версия TensorFlow: {tf.__version__}")

print(" Библиотека keras поможет создать нейронную сеть")

print(" Библиотека matplotlib поможет построить графики")

Ячейка 3: Генерация синтетических данных химического процесса

Создайте новую ячейку с кодом и скопируйте:

===== ГЕНЕРАЦИЯ ДАННЫХ =====

Создаем искусственные данные для обучения нейросети

Фиксируем случай для воспроизводимости результатов

np.random.seed(42)

Количество экспериментальных точек

n_samples = 100

===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: задать диапазоны параметров процесса согласно варианту

Вариант 1: T=[50;120]°C, P=[1;5]атм

Вариант 2: T=[30;100]°C, P=[2;8]атм

Вариант 3: T=[40;110]°C, P=[1.5;6]атм

Ваш код ниже (пример для Варианта 1):

```

T_min, T_max = 50, 120    # Температура, °C
P_min, P_max = 1, 5      # Давление, атм

# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====

# Генерируем входные параметры
T = np.random.uniform(T_min, T_max, n_samples)
P = np.random.uniform(P_min, P_max, n_samples)

# Целевая переменная: выход продукта (%) с нелинейной зависимостью и шумом
Yield = 0.4*T + 1.2*P - 0.01*T*P + np.random.normal(0, 3, n_samples)
Yield = np.clip(Yield, 0, 100) # Ограничиваем выход в физически допустимых пределах

# Формируем матрицу признаков и вектор цели
X = np.column_stack([T, P])
y = Yield

print(f"    Сгенерировано {n_samples} экспериментальных точек")
print(f"    Диапазон выхода продукта: {y.min():.1f} – {y.max():.1f} %")

```

Ячейка 4: Визуализация исходных данных

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ВИЗУАЛИЗАЦИЯ ДАННЫХ =====
```

```

plt.figure(figsize=(12, 5))

# График 1: Зависимость выхода от температуры
plt.subplot(1, 2, 1)
plt.scatter(T, y, c=P, cmap='viridis', alpha=0.7)
plt.xlabel('Температура, °C')
plt.ylabel('Выход продукта, %')
plt.title('Выход vs Температура (цвет = давление)')
plt.colorbar(label='Давление, атм')
plt.grid(True, alpha=0.3)

# График 2: Зависимость выхода от давления
plt.subplot(1, 2, 2)
plt.scatter(P, y, c=T, cmap='plasma', alpha=0.7)
plt.xlabel('Давление, атм')
plt.ylabel('Выход продукта, %')
plt.title('Выход vs Давление (цвет = температура)')
plt.colorbar(label='Температура, °C')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("    Графики построены. Проанализируйте характер зависимостей.")

```

Ячейка 5: Создание архитектуры перцептрона (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 1: СОЗДАНИЕ МОДЕЛИ =====

```
def create_perceptron(input_dim=2):
    """
    Создает простейшую нейронную сеть (перцептрон) для регрессии.
    input_dim: количество входных признаков (по умолчанию 2: Т и Р)
    """
    # ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
    # ЗАДАНИЕ: создать модель Sequential с одним выходным нейроном

    # Подсказка:
    # - Используйте Dense(1, activation='linear') для задачи регрессии
    # - input_shape=(input_dim,) задает размерность входа

    # Ваш код ниже:

    model = Sequential([
        Dense(1, activation='linear', input_shape=(input_dim,))
    ])

    # ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

    return model

# Создаем модель
model = create_perceptron(input_dim=2)
model.summary()

print("    Модель перцептрона создана")
```

Ячейка 6: Компиляция и обучение модели (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 2: ОБУЧЕНИЕ МОДЕЛИ =====

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: скомпилировать модель и запустить обучение

# Подсказка:
# - optimizer='adam' – адаптивный оптимизатор
# - loss='mse' – среднеквадратичная ошибка для регрессии
# - epochs=100 – количество проходов по данным
# - verbose=0 – скрыть подробный вывод при обучении

# Ваш код ниже:

# Компиляция модели
model.compile(optimizer='adam', loss='mse')

# Обучение модели
history = model.fit(X, y, epochs=100, batch_size=16, verbose=0)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
print("    Обучение завершено")
print(f"    Финальное значение потери (MSE): {history.history['loss'][-1]:.3f}")
```

Ячейка 7: Визуализация процесса обучения (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 3: АНАЛИЗ СХОДИМОСТИ =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: построить график изменения ошибки в процессе обучения
```

```
# Подсказка: используйте history.history['loss'] и plt.plot()
```

```
# Ваш код ниже:
```

```
plt.figure(figsize=(8, 4))
plt.plot(history.history['loss'], linewidth=2)
plt.xlabel('Эпоха')
plt.ylabel('MSE (ошибка)')
plt.title('Динамика обучения перцептрона')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
print("    График обучения построен")
```

Ячейка 8: Прогнозирование и оценка качества (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 4: ОЦЕНКА КАЧЕСТВА =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: получить предсказания модели и рассчитать метрики
```

```
# Подсказка:
```

```
# - model.predict(X) возвращает прогнозы
```

```
# - mean_squared_error(y_true, y_pred) из sklearn считает MSE
```

```
# - r2_score(y_true, y_pred) считает коэффициент детерминации
```

```
# Ваш код ниже:
```

```
# Получаем предсказания
```

```
y_pred = model.predict(X, verbose=0).flatten()
```

```
# Рассчитываем метрики
```

```
mse = mean_squared_error(y, y_pred)
```

```
r2 = r2_score(y, y_pred)
```

```
print(f"    Метрики качества модели:")
```

```
print(f"    MSE = {mse:.3f} (%²)")
```

```
print(f" RMSE = {np.sqrt(mse):.3f} (%)")
print(f" R2 = {r2:.3f}")
```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 9: Визуализация результатов прогнозирования

Создайте новую ячейку с кодом и скопируйте:

===== ВИЗУАЛИЗАЦИЯ ПРОГНОЗОВ =====

```
plt.figure(figsize=(14, 5))
```

```
# График 1: Факт vs Прогноз
```

```
plt.subplot(1, 2, 1)
```

```
plt.scatter(y, y_pred, alpha=0.7, edgecolors='black')
```

```
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', linewidth=2)
```

```
plt.xlabel('Фактический выход, %')
```

```
plt.ylabel('Прогнозируемый выход, %')
```

```
plt.title('Факт против Прогноза')
```

```
plt.grid(True, alpha=0.3)
```

```
# График 2: Остатки модели
```

```
plt.subplot(1, 2, 2)
```

```
residuals = y - y_pred
```

```
plt.scatter(y_pred, residuals, alpha=0.7, edgecolors='black')
```

```
plt.axhline(0, color='red', linestyle='--', linewidth=1)
```

```
plt.xlabel('Прогнозируемый выход, %')
```

```
plt.ylabel('Остаток (факт - прогноз), %')
```

```
plt.title('Анализ остатков')
```

```
plt.grid(True, alpha=0.3)
```

```
plt.tight_layout()
```

```
plt.show()
```

```
print(" Графики прогнозирования построены")
```

```
print(" Идеальная модель: точки на красной линии (график 1), остатки вокруг нуля (график 2)")
```

Ячейка 10: Прогноз для новых условий процесса (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: ПРОГНОЗ ДЛЯ НОВЫХ УСЛОВИЙ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: спрогнозировать выход продукта для заданных параметров

Подсказка:

- Создайте массив новых входных данных: np.array([[T_new, P_new]])

- Используйте model.predict() для получения прогноза

Ваш код ниже (пример):

Параметры нового эксперимента (измените согласно варианту!)

T_new = 85 # Температура, °C

```

P_new = 3.2 # Давление, атм

# Формируем входной вектор
X_new = np.array([[T_new, P_new]])

# Получаем прогноз
yield_pred = model.predict(X_new, verbose=0)[0, 0]

print(f" Прогноз для новых условий:")
print(f" T = {T_new} °C, P = {P_new} атм")
print(f" → Ожидаемый выход продукта: {yield_pred:.2f} %")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 11: Сохранение результатов и модели

Создайте новую ячейку с кодом и скопируйте:

```

# ===== СОХРАНЕНИЕ РЕЗУЛЬТАТОВ =====

from datetime import datetime

# Формируем отчет в виде словаря
results = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'samples': n_samples,
    'T_range': f'{T_min}-{T_max} °C',
    'P_range': f'{P_min}-{P_max} атм',
    'mse': float(mse),
    'rmse': float(np.sqrt(mse)),
    'r2': float(r2),
    'prediction_T': T_new,
    'prediction_P': P_new,
    'prediction_yield': float(yield_pred)
}

# Вывод сводки на экран
print("\n" + "="*60)
print(" СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №1")
print("="*60)
for key, value in results.items():
    print(f"{key:20s}: {value}")
print("="*60)

# Сохранение модели
model.save('perceptron_chem_model.keras')
print(" Модель сохранена в файл: perceptron_chem_model.keras")

# Скачивание отчета (опционально)
# from google.colab import files
# import json

```

```
# with open('lab1_results.json', 'w', encoding='utf-8') as f:
#     json.dump(results, f, ensure_ascii=False, indent=2)
# files.download('lab1_results.json')
```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность обучения

1. Запустите все ячейки последовательно (Shift+Enter)
2. Убедитесь, что значение $R^2 > 0.7$ (модель адекватно описывает данные)
3. Проверьте, что график «Факт vs Прогноз» показывает близость точек к диагонали

Тест 2: Изменение варианта

1. В Ячейке 3 измените диапазоны T_min, T_max, P_min, P_max согласно своему варианту
2. Перезапустите все ячейки (Меню → Выполнить все)
3. Зафиксируйте, как изменились метрики качества

Тест 3: Прогноз для новых условий

1. В Ячейке 10 задайте свои значения T_new, P_new
2. Убедитесь, что прогноз попадает в физически допустимый диапазон (0–100%)
3. Прокомментируйте результат: соответствует ли он ожидаемой тенденции?

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Что должен содержать отчет (в формате Word):

Структура отчета:

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер лабораторной работы)
2. **Тема и цель работы** (кратко, 2–3 предложения)
3. **Условия задания** (номер варианта, диапазоны параметров, архитектура сети)
4. **Разработанные документы:**
 - Скриншоты ключевых графиков (исходные данные, динамика обучения, факт/прогноз, остатки)
 - Таблица с рассчитанными метриками (MSE, RMSE, R^2)
 - Результат прогноза для новых условий
5. **Выводы** (3–4 пункта: что освоено, какая точность достигнута, перспективы применения)
6. **Приложение** (исходный код ноутбука или ссылка на Google Colab)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Что такое перцептрон и чем он отличается от многослойной нейронной сети? В каких случаях достаточно использовать перцептрон для моделирования химико-технологических процессов?

Вопрос 2:

Почему для задачи прогнозирования выхода продукта используется функция активации linear на выходном слое, а не sigmoid или relu?

Вопрос 3:

Что показывает коэффициент детерминации R^2 ? Как интерпретировать значение $R^2 = 0.85$ в контексте химического эксперимента?

Вопрос 4:

Почему важно визуализировать остатки модели (разность «факт – прогноз»)? О чем свидетельствует систематическое отклонение остатков от нуля?

Вопрос 5:

Какие ограничения имеет синтетическая генерация данных для обучения нейросети? Что

необходимо изменить в программе, чтобы работать с реальными экспериментальными данными?

Лабораторная работа №2

«Однослойная нейронная сеть для регрессии: моделирование нелинейной зависимости выхода продукта»

Цель: Создать программу для обучения нейронной сети с одним скрытым слоем, способной аппроксимировать нелинейную зависимость выхода химической реакции от технологических параметров, и сравнить ее точность с линейной моделью.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 2: Однослойная сеть для регрессии

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №2
```

```
## Однослойная нейронная сеть для регрессии
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует данные с нелинейной зависимостью (выход продукта от T и P)
2. Строит нейронную сеть с одним скрытым слоем
3. Обучает сеть и сравнивает с линейной моделью
4. Визуализирует преимущества нелинейной аппроксимации
5. Оценивает качество по метрикам MSE и R^2

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error, r2_score
```

```
print(" Библиотеки загружены!")
```

```
print(f"Версия TensorFlow: {tf.__version__}")
```

```
print(" Keras: создание нейронных сетей")
```

```
print(" sklearn: линейная регрессия для сравнения")
print(" matplotlib: визуализация результатов")
```

Ячейка 3: Генерация нелинейных данных химического процесса

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ГЕНЕРАЦИЯ НЕЛИНЕЙНЫХ ДАННЫХ =====
```

```
np.random.seed(42)
n_samples = 150
```

```
# ===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: задать параметры процесса и тип нелинейности
```

```
# Вариант 1: квадратичная зависимость от T
# Вариант 2: экспоненциальный рост с насыщением
# Вариант 3: произведение T*P с квадратичным членом
```

```
# Ваш код ниже (пример для Варианта 1):
T_min, T_max = 40, 110 # Температура, °C
P_min, P_max = 1.5, 6.0 # Давление, атм
nonlinearity_type = 'quadratic' # 'quadratic', 'exponential', 'interaction'
```

```
# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====
```

```
# Генерация входных данных
T = np.random.uniform(T_min, T_max, n_samples)
P = np.random.uniform(P_min, P_max, n_samples)
```

```
# Нелинейная целевая функция
if nonlinearity_type == 'quadratic':
    Yield = 0.3*T + 1.0*P - 0.002*T**2 + np.random.normal(0, 2.5, n_samples)
elif nonlinearity_type == 'exponential':
    Yield = 50 * (1 - np.exp(-0.03*(T-40))) + 0.8*P + np.random.normal(0, 2.5, n_samples)
else: # interaction
    Yield = 0.2*T + 0.9*P + 0.005*T*P - 0.001*T**2 + np.random.normal(0, 2.5, n_samples)
```

```
Yield = np.clip(Yield, 0, 100)
```

```
X = np.column_stack([T, P])
y = Yield
```

```
print(f" Сгенерировано {n_samples} точек с нелинейностью '{nonlinearity_type}'")
print(f" Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")
print(f" Коэффициент корреляции T-Y: {np.corrcoef(T, y)[0,1]:.3f}")
```

Ячейка 4: Визуализация нелинейной зависимости

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ВИЗУАЛИЗАЦИЯ НЕЛИНЕЙНОСТИ =====
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```

# График 1: 3D-подобная визуализация (цвет = выход)
sc1 = axes[0].scatter(T, P, c=y, cmap='viridis', s=50, edgcolors='black')
axes[0].set_xlabel('Температура, °C')
axes[0].set_ylabel('Давление, атм')
axes[0].set_title('Параметры процесса (цвет = выход продукта)')
plt.colorbar(sc1, ax=axes[0], label='Выход, %')
axes[0].grid(True, alpha=0.3)

# График 2: Зависимость выхода от температуры (с группировкой по давлению)
for P_level, color in zip([2, 4, 6], ['blue', 'green', 'red']):
    mask = (P >= P_level-0.5) & (P < P_level+0.5)
    if mask.sum() > 0:
        axes[1].scatter(T[mask], y[mask], c=color, label=f'P ≈ {P_level} атм', alpha=0.7, s=30)

axes[1].set_xlabel('Температура, °C')
axes[1].set_ylabel('Выход продукта, %')
axes[1].set_title('Выход vs Температура (разные уровни давления)')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("    Графики построены. Обратите внимание на нелинейный характер зависимостей.")

```

Ячейка 5: Создание нейронной сети с одним скрытым слоем (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 1: АРХИТЕКТУРА СЕТИ =====

```

def create_single_hidden_layer_nn(input_dim=2, hidden_units=8):
    """
    Создает нейронную сеть с одним скрытым слоем для регрессии.
    input_dim: количество входных признаков
    hidden_units: количество нейронов в скрытом слое
    """
    # ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
    # ЗАДАНИЕ: создать модель с одним скрытым слоем (ReLU) и линейным выходом

    # Подсказка:
    # - Первый Dense: hidden_units нейронов, activation='relu', input_shape=(input_dim,)
    # - Второй Dense: 1 нейрон, activation='linear' (для регрессии)

    # Ваш код ниже:

    model = Sequential([
        Dense(hidden_units, activation='relu', input_shape=(input_dim,)),
        Dense(1, activation='linear')
    ])

    # ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

    return model

```

```
# ===== ВАШ ВАРИАНТ: выбор количества нейронов =====
# Вариант 1: 4 нейрона | Вариант 2: 8 нейронов | Вариант 3: 12 нейронов
hidden_neurons = 8 # Измените согласно варианту!

# Создаем модель
model_nn = create_single_hidden_layer_nn(input_dim=2, hidden_units=hidden_neurons)
model_nn.summary()

print(f"    Создана сеть с {hidden_neurons} нейронами в скрытом слое")
```

Ячейка 6: Обучение нейронной сети и линейной модели для сравнения (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 2: ОБУЧЕНИЕ И СРАВНЕНИЕ =====
```

```
# ===== ЧАСТЬ А: Обучение нейронной сети =====
```

```
print("    Обучение нейронной сети...")
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: скомпилировать и обучить нейронную сеть
```

```
model_nn.compile(optimizer='adam', loss='mse')
history_nn = model_nn.fit(X, y, epochs=150, batch_size=16, verbose=0)
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
# ===== ЧАСТЬ Б: Обучение линейной регрессии для сравнения =====
```

```
print("    Обучение линейной регрессии (для сравнения)...")
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: обучить линейную модель sklearn и получить предсказания
```

```
lin_reg = LinearRegression()
lin_reg.fit(X, y)
y_pred_lin = lin_reg.predict(X)
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
# Предсказания нейросети
```

```
y_pred_nn = model_nn.predict(X, verbose=0).flatten()
```

```
print("    Обучение завершено")
```

Ячейка 7: Сравнение метрик качества (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 3: СРАВНЕНИЕ МЕТОК =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: рассчитать и сравнить метрики для обеих моделей
```

```
# Ваш код ниже:
```

```

# Метрики для нейронной сети
mse_nn = mean_squared_error(y, y_pred_nn)
r2_nn = r2_score(y, y_pred_nn)

# Метрики для линейной регрессии
mse_lin = mean_squared_error(y, y_pred_lin)
r2_lin = r2_score(y, y_pred_lin)

# Вывод сравнения
print("\n" + "="*60)
print("    СРАВНЕНИЕ КАЧЕСТВА МОДЕЛЕЙ")
print("="*60)
print(f"{'Метрика':<15} {'Линейная':<20} {'Нейросеть':<20}")
print("-"*60)
print(f"{'MSE':<15} {mse_lin:<20.3f} {mse_nn:<20.3f}")
print(f"{'RMSE':<15} {np.sqrt(mse_lin):<20.3f} {np.sqrt(mse_nn):<20.3f}")
print(f"{'R²':<15} {r2_lin:<20.3f} {r2_nn:<20.3f}")
print("="*60)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

# Визуализация сравнения
plt.figure(figsize=(8, 4))
models = ['Линейная', 'Нейросеть']
r2_values = [r2_lin, r2_nn]
colors = ['skyblue', 'lightcoral']

bars = plt.bar(models, r2_values, color=colors, edgecolor='black')
plt.ylabel('R² (коэффициент детерминации)')
plt.title('Сравнение качества аппроксимации')
plt.ylim(0, 1.1)

# Добавление значений на столбцы
for bar, val in zip(bars, r2_values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
             f'{val:.3f}', ha='center', va='bottom', fontsize=10)

plt.grid(True, alpha=0.3, axis='y')
plt.show()

print(f"    Преимущество нейросети:  $\Delta R^2 = \{r2\_nn - r2\_lin:+.3f\}$ ")

```

Ячейка 8: Визуализация аппроксимации нелинейной зависимости (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 4: ВИЗУАЛИЗАЦИЯ АППРОКСИМАЦИИ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: построить графики "Факт vs Прогноз" для обеих моделей

Ваш код ниже:

```

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# График для линейной модели
axes[0].scatter(y, y_pred_lin, alpha=0.6, edgecolors='black', label='Данные')
axes[0].plot([y.min(), y.max()], [y.min(), y.max()], 'r--', linewidth=2, label='Идеальный прогноз')
axes[0].set_xlabel('Фактический выход, %')
axes[0].set_ylabel('Прогноз (линейная модель), %')
axes[0].set_title(f'Линейная регрессия ( $R^2 = \{r2\_lin:.3f\}$ )')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# График для нейронной сети
axes[1].scatter(y, y_pred_nn, alpha=0.6, edgecolors='black', label='Данные', color='green')
axes[1].plot([y.min(), y.max()], [y.min(), y.max()], 'r--', linewidth=2, label='Идеальный прогноз')
axes[1].set_xlabel('Фактический выход, %')
axes[1].set_ylabel('Прогноз (нейросеть), %')
axes[1].set_title(f'Нейронная сеть ( $R^2 = \{r2\_nn:.3f\}$ )')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print("    Графики построены. Сравните рассеяние точек относительно красной линии.")

```

Ячейка 9: Анализ остатков моделей (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: АНАЛИЗ ОСТАТКОВ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: построить графики остатков и выявить систематические ошибки

Ваш код ниже:

```

residuals_lin = y - y_pred_lin
residuals_nn = y - y_pred_nn

```

```

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

```

Остатки линейной модели

```

axes[0].scatter(y_pred_lin, residuals_lin, alpha=0.6, edgecolors='black')
axes[0].axhline(0, color='red', linestyle='--', linewidth=1)
axes[0].set_xlabel('Прогноз (линейная)')
axes[0].set_ylabel('Остаток (факт - прогноз)')
axes[0].set_title('Остатки линейной модели')
axes[0].grid(True, alpha=0.3)

```

Остатки нейронной сети

```

axes[1].scatter(y_pred_nn, residuals_nn, alpha=0.6, edgecolors='black', color='green')
axes[1].axhline(0, color='red', linestyle='--', linewidth=1)
axes[1].set_xlabel('Прогноз (нейросеть)')
axes[1].set_ylabel('Остаток (факт - прогноз)')
axes[1].set_title('Остатки нейронной сети')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Статистика остатков

```

print("\n    Статистика остатков:")
print(f"Линейная: mean = {residuals_lin.mean():+.3f}, std = {residuals_lin.std():.3f}")
print(f"Нейросеть: mean = {residuals_nn.mean():+.3f}, std = {residuals_nn.std():.3f}")
print("    Идеальные остатки: среднее  $\approx 0$ , случайное распределение вокруг нуля")

```

Ячейка 10: Прогноз для новых условий и анализ чувствительности

Создайте новую ячейку с кодом и скопируйте:

===== ПРОГНОЗ И АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ =====

===== ВАШ ВАРИАНТ: параметры для прогноза =====

Измените значения согласно заданию!

T_new = 75 # Температура, °C

P_new = 4.0 # Давление, атм

Прогноз нейросети

```

X_new = np.array([[T_new, P_new]])
yield_pred_nn = model_nn.predict(X_new, verbose=0)[0, 0]
yield_pred_lin = lin_reg.predict(X_new)[0]

```

```

print(f"    Прогноз для новых условий (T={T_new}°C, P={P_new} атм):")
print(f"    Линейная модель: {yield_pred_lin:.2f} %")
print(f"    Нейронная сеть: {yield_pred_nn:.2f} %")
print(f"    Разница: {abs(yield_pred_nn - yield_pred_lin):.2f} %")

```

===== ВАША РАБОТА: анализ чувствительности =====

ЗАДАНИЕ: исследовать, как меняется прогноз при изменении одного параметра

```

print("\n    Анализ чувствительности к температуре:")
T_range = np.linspace(T_min, T_max, 20)
P_fixed = P_new

```

```

predictions = []
for T_val in T_range:
    X_test = np.array([[T_val, P_fixed]])
    pred = model_nn.predict(X_test, verbose=0)[0, 0]
    predictions.append(pred)

```

Визуализация

```

plt.figure(figsize=(8, 5))
plt.plot(T_range, predictions, linewidth=2, color='green', label='Прогноз нейросети')
plt.scatter([T_new], [yield_pred_nn], color='red', s=100, zorder=5, label='Ваш прогноз')
plt.xlabel('Температура, °C')
plt.ylabel('Прогнозируемый выход, %')
plt.title(f'Зависимость выхода от температуры (P = {P_fixed} атм)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()

print("    График чувствительности построен")

```

Ячейка 11: Сохранение результатов и отчета

Создайте новую ячейку с кодом и скопируйте:

===== СОХРАНЕНИЕ РЕЗУЛЬТАТОВ =====

```

from datetime import datetime

# Формируем сводку
results = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'nonlinearity': nonlinearity_type,
    'hidden_neurons': hidden_neurons,
    'linear_R2': float(r2_lin),
    'nn_R2': float(r2_nn),
    'improvement': float(r2_nn - r2_lin),
    'prediction_T': T_new,
    'prediction_P': P_new,
    'prediction_linear': float(yield_pred_lin),
    'prediction_nn': float(yield_pred_nn)
}

# Вывод сводки
print("\n" + "="*60)
print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №2")
print("="*60)
for key, value in results.items():
    print(f"{key:20s}: {value}")
print("="*60)

# Сохранение моделей
model_nn.save('nn_regression_model.keras')
print("    Нейросеть сохранена: nn_regression_model.keras")

# Опционально: скачивание результатов
# from google.colab import files
# import json
# with open('lab2_results.json', 'w', encoding='utf-8') as f:
#     json.dump(results, f, ensure_ascii=False, indent=2)

```

```
# files.download('lab2_results.json')
```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность обучения

1. Запустите все ячейки последовательно
2. Убедитесь, что R^2 нейросети $>$ R^2 линейной модели (преимущество нелинейной аппроксимации)
3. Проверьте, что остатки нейросети распределены случайнее, чем у линейной модели

Тест 2: Влияние количества нейронов

1. В Ячейке 5 измените `hidden_neurons` ($4 \rightarrow 8 \rightarrow 12$)
2. Перезапустите обучение и сравните метрики
3. Зафиксируйте, при каком числе нейронов достигается оптимальное качество

Тест 3: Анализ чувствительности

1. В Ячейке 10 задайте свои значения T_{new} , P_{new}
2. Постройте график зависимости выхода от температуры
3. Прокомментируйте физическую интерпретацию полученной кривой

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, тип нелинейности, архитектура сети)
4. **Разработанные документы**:
 - Скриншоты: визуализация данных, сравнение моделей, графики остатков, анализ чувствительности
 - Таблица сравнения метрик (MSE, RMSE, R^2 для обеих моделей)
 - Результат прогноза для новых условий с комментарием
5. **Выводы** (3–4 пункта):
 - В чем проявилось преимущество нейросети перед линейной моделью?
 - Как количество нейронов влияет на качество аппроксимации?
 - Перспективы применения подхода для реальных химических процессов
6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему нейронная сеть с одним скрытым слоем и функцией активации ReLU способна аппроксимировать нелинейные зависимости, а линейная регрессия – нет?

Вопрос 2:

Что означает ситуация, когда R^2 нейросети $<$ R^2 линейной модели? Какие возможные причины этого явления?

Вопрос 3:

Как по графику остатков можно определить, что модель не учитывает какую-либо закономерность в данных? Приведите пример из выполненной работы.

Вопрос 4:

Почему для задачи регрессии на выходном слое используется линейная активация, даже если в скрытом слое применяется ReLU?

Вопрос 5:

Какие ограничения имеет использование синтетических данных для обучения? Что необходимо изменить в программе, чтобы подготовить ее к работе с реальными экспериментальными данными химического процесса?

Лабораторная работа №3

«Многослойный перцептрон (MLP): моделирование сложных нелинейных зависимостей в химических процессах»

Цель: Создать программу для обучения нейронной сети с двумя скрытыми слоями, способной аппроксимировать сложные многофакторные зависимости химико-технологических процессов, и освоить методику разделения данных на обучающую и тестовую выборки для корректной оценки качества модели.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 3: Многослойный перцептрон

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №3
```

```
## Многослойный перцептрон для моделирования химических процессов
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует многофакторные данные химического процесса (3+ параметра)
2. Разделяет данные на обучающую и тестовую выборки
3. Строит нейронную сеть с двумя скрытыми слоями (MLP)
4. Обучает сеть и оценивает качество на тестовых данных
5. Сравнивает результаты с моделями из предыдущих работ

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```

print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras: создание многослойных сетей")
print(" sklearn: предобработка данных и метрики")
print(" matplotlib: визуализация результатов")

```

Ячейка 3: Генерация многофакторных данных химического процесса

Создайте новую ячейку с кодом и скопируйте:

===== ГЕНЕРАЦИЯ МНОГОФАКТОРНЫХ ДАННЫХ =====

```

np.random.seed(42)
n_samples = 300

```

===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: задать параметры процесса согласно варианту

Вариант 1: 3 фактора (Т, Р, концентрация катализатора)

Вариант 2: 4 фактора (Т, Р, С, время реакции)

Вариант 3: 3 фактора с взаимодействием (Т, Р, рН)

num_features = 3 # Измените согласно варианту!

Диапазоны параметров

T_min, T_max = 30, 120 # Температура, °C

P_min, P_max = 1, 8 # Давление, атм

C_min, C_max = 0.1, 2.5 # Концентрация катализатора, моль/л

===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====

Генерация входных данных

T = np.random.uniform(T_min, T_max, n_samples)

P = np.random.uniform(P_min, P_max, n_samples)

C = np.random.uniform(C_min, C_max, n_samples)

Сложная нелинейная целевая функция с взаимодействиями

if num_features == 3:

Yield = (0.25*T + 0.8*P + 15*C -
0.0015*T**2 + 0.05*P*C - 0.003*T*C +
np.random.normal(0, 2.0, n_samples))

else: # 4 фактора

tau = np.random.uniform(10, 120, n_samples) # время, мин

Yield = (0.2*T + 0.7*P + 12*C + 0.1*tau -
0.001*T**2 + 0.03*P*C + 0.002*T*tau +
np.random.normal(0, 2.0, n_samples))

X_extra = tau.reshape(-1, 1)

Yield = np.clip(Yield, 0, 100)

Формирование матрицы признаков

if num_features == 3:

X = np.column_stack([T, P, C])

```

feature_names = ['T_C', 'P_atm', 'Catalyst_molL']
else:
    X = np.column_stack([T, P, C, tau])
    feature_names = ['T_C', 'P_atm', 'Catalyst_molL', 'tau_min']

y = Yield

```

```

print(f"    Сгенерировано {n_samples} точек с {num_features} факторами")
print(f"    Признаки: {feature_names}")
print(f"    Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")

```

Ячейка 4: Разделение данных и стандартизация (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 1: ПОДГОТОВКА ДАННЫХ =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: разделить данные на обучающую и тестовую выборки, выполнить
стандартизацию
```

```
# Подсказка:
```

```
# - train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# - StandardScaler().fit_transform() применяется только к обучающей выборке
```

```
# - К тестовой применяется только transform()
```

```
# Ваш код ниже:
```

```
# Разделение данных
```

```

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

```
print(f"    Обучающая выборка: {X_train.shape[0]} образцов")
```

```
print(f"    Тестовая выборка: {X_test.shape[0]} образцов")
```

```
# Стандартизация признаков
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
print("    Стандартизация выполнена (mean≈0, std≈1)")
```

```
print(f"    Проверка: mean={X_train_scaled.mean():.3f}, std={X_train_scaled.std():.3f}")
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

Ячейка 5: Создание архитектуры MLP с двумя скрытыми слоями (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 2: АРХИТЕКТУРА МНОГОСЛОЙНОЙ СЕТИ
```

```
=====
```

```
def create_mlp(input_dim, layer1_units=16, layer2_units=8):
```

```

"""
Создает многослойный перцептрон с двумя скрытыми слоями.
"""

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: создать модель Sequential с двумя скрытыми слоями (ReLU) и линейным
ВЫХОДОМ

# Подсказка:
# - Первый Dense: layer1_units нейронов, activation='relu', input_shape=(input_dim,)
# - Второй Dense: layer2_units нейронов, activation='relu'
# - Выходной Dense: 1 нейрон, activation='linear'

# Ваш код ниже:

model = Sequential([
    Dense(layer1_units, activation='relu', input_shape=(input_dim,)),
    Dense(layer2_units, activation='relu'),
    Dense(1, activation='linear')
])

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

return model

# ===== ВАШ ВАРИАНТ: архитектура сети =====
# Вариант 1: [8, 4] нейронов | Вариант 2: [16, 8] | Вариант 3: [12, 6]
units_layer1 = 16 # Измените согласно варианту!
units_layer2 = 8 # Измените согласно варианту!

# Создаем модель
model_mlp = create_mlp(input_dim=X_train_scaled.shape[1],
                        layer1_units=units_layer1,
                        layer2_units=units_layer2)
model_mlp.summary()

print(f" Создан MLP с архитектурой: {units_layer1} → {units_layer2} → 1")

```

Ячейка 6: Обучение модели с использованием EarlyStopping (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ЗАДАНИЕ 3: ОБУЧЕНИЕ С КОНТРОЛЕМ ПЕРЕОБУЧЕНИЯ
=====

```

```

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

```

```

# ЗАДАНИЕ: скомпилировать модель и обучить с использованием EarlyStopping

```

```

# Подсказка:

```

```

# - optimizer='adam', loss='mse', metrics=['mae']

```

```

# - EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)

```

```

# - validation_split=0.2 при обучении

```

```

# Ваш код ниже:

```

```

# Компиляция модели
model_mlp.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Callback для ранней остановки
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True,
    verbose=0
)

# Обучение модели
history = model_mlp.fit(
    X_train_scaled, y_train,
    epochs=200,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print(f"    Обучение завершено на эпохе {len(history.history['loss'])}")
print(f"    Финальная ошибка: train_loss={history.history['loss'][-1]:.3f},
val_loss={history.history['val_loss'][-1]:.3f}")

```

Ячейка 7: Визуализация процесса обучения (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 4: АНАЛИЗ ДИНАМИКИ ОБУЧЕНИЯ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: построить графики изменения потерь и метрик в процессе обучения

Ваш код ниже:

```

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# График потерь (MSE)
axes[0].plot(history.history['loss'], label='Train', linewidth=2)
axes[0].plot(history.history['val_loss'], label='Validation', linewidth=2)
axes[0].set_xlabel('Эпоха')
axes[0].set_ylabel('MSE (потеря)')
axes[0].set_title('Динамика ошибки в процессе обучения')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# График MAE
axes[1].plot(history.history['mae'], label='Train MAE', linewidth=2, color='orange')
axes[1].plot(history.history['val_mae'], label='Val MAE', linewidth=2, color='red')
axes[1].set_xlabel('Эпоха')
axes[1].set_ylabel('MAE (средняя абсолютная ошибка)')

```

```
axes[1].set_title("Точность прогноза в единицах выхода (%)")
axes[1].legend()
axes[1].grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
print("    Графики обучения построены")
print("    EarlyStopping остановил обучение, когда валидационная ошибка перестала
улучшаться")
```

Ячейка 8: Оценка качества на тестовой выборке (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 5: ОЦЕНКА НА НЕЗАВИСИМЫХ ДАННЫХ
=====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: получить предсказания на тестовой выборке и рассчитать метрики
```

```
# Ваш код ниже:
```

```
# Прогноз на тестовых данных
y_test_pred = model_mlp.predict(X_test_scaled, verbose=0).flatten()
```

```
# Расчет метрик
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)
mae_test = mean_absolute_error(y_test, y_test_pred)
r2_test = r2_score(y_test, y_test_pred)
```

```
# Вывод результатов
print("\n" + "="*60)
print("    КАЧЕСТВО МОДЕЛИ НА ТЕСТОВОЙ ВЫБОРКЕ")
print("="*60)
print(f"{'Метрика':<25} {'Значение':>15}")
print("-"*60)
print(f"{'MSE':<25} {mse_test:>15.3f}")
print(f"{'RMSE':<25} {rmse_test:>15.3f} %")
print(f"{'MAE':<25} {mae_test:>15.3f} %")
print(f"{'R² (коэфф. детерминации)':<25} {r2_test:>15.3f}")
print("="*60)
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```
# Интерпретация
if r2_test > 0.9:
    print("    Отличное качество: модель объясняет >90% дисперсии данных")
elif r2_test > 0.75:
    print("    Хорошее качество: модель пригодна для прогнозирования")
```

else:

```
print("    Умеренное качество: требуется доработка архитектуры или данных")
```

Ячейка 9: Визуализация результатов прогнозирования

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ВИЗУАЛИЗАЦИЯ ПРОГНОЗОВ =====
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```
# График 1: Факт против Прогноза
```

```
axes[0].scatter(y_test, y_test_pred, alpha=0.7, edgecolors='black', s=40)
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
axes[0].set_xlabel('Фактический выход, %')
axes[0].set_ylabel('Прогнозируемый выход, %')
axes[0].set_title(f'Тестовая выборка: Факт vs Прогноз ( $R^2 = \{r2\_test:.3f\}$ )')
axes[0].grid(True, alpha=0.3)
```

```
# График 2: Остатки модели
```

```
residuals = y_test - y_test_pred
axes[1].scatter(y_test_pred, residuals, alpha=0.7, edgecolors='black', s=40)
axes[1].axhline(0, color='red', linestyle='--', linewidth=1)
axes[1].set_xlabel('Прогнозируемый выход, %')
axes[1].set_ylabel('Остаток (факт - прогноз), %')
axes[1].set_title('Анализ остатков на тестовых данных')
axes[1].grid(True, alpha=0.3)
```

```
plt.tight_layout()
```

```
plt.show()
```

```
print("    Графики прогнозирования построены")
```

```
print("    Идеальная модель: точки на диагонали (график 1), остатки  $\sim N(0, \sigma)$  (график 2)")
```

Ячейка 10: Прогноз для новых технологических условий

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ПРОГНОЗ ДЛЯ НОВЫХ УСЛОВИЙ ПРОЦЕССА =====
```

```
# ===== ВАШ ВАРИАНТ: параметры для прогноза =====
```

```
# Измените значения согласно заданию!
```

```
T_new = 80      # Температура, °C
```

```
P_new = 4.5     # Давление, атм
```

```
C_new = 1.8     # Концентрация катализатора, моль/л
```

```
# Формирование входного вектора
```

```
if num_features == 3:
```

```
    X_new = np.array([[T_new, P_new, C_new]])
```

```
else:
```

```
    tau_new = 60 # время реакции, мин (для 4-факторной модели)
```

```
    X_new = np.array([[T_new, P_new, C_new, tau_new]])
```

```
# Стандартизация новых данных (ВАЖНО: использовать тот же scaler!)
```

```
X_new_scaled = scaler.transform(X_new)
```

```

# Прогноз
yield_pred = model_mlp.predict(X_new_scaled, verbose=0)[0, 0]

print(f"    Прогноз для новых условий:")
if num_features == 3:
    print(f"    T = {T_new} °C, P = {P_new} атм, C = {C_new} моль/л")
else:
    print(f"    T = {T_new} °C, P = {P_new} атм, C = {C_new} моль/л, τ = {tau_new} мин")
print(f"    → Ожидаемый выход продукта: {yield_pred:.2f} %")

# Проверка физической осмысленности
if 0 <= yield_pred <= 100:
    print("    Прогноз в физически допустимом диапазоне")
else:
    print("    Прогноз выходит за физические границы – модель требует доработки")

```

Ячейка 11: Сравнение с предыдущими моделями и сохранение результатов

Создайте новую ячейку с кодом и скопируйте:

===== СРАВНЕНИЕ МОДЕЛЕЙ И СОХРАНЕНИЕ =====

```

from datetime import datetime

```

===== ВАША РАБОТА: сравнительная таблица =====

ЗАДАНИЕ: заполнить метрики для моделей из лабораторных работ №1 и №2

(значения взять из отчетов предыдущих работ)

Пример заполнения (замените на свои значения!):

```

comparison = {
    'Perceptron (Lab1)': {'R2_train': 0.72, 'RMSE': 4.8},
    'Single Hidden (Lab2)': {'R2_train': 0.85, 'RMSE': 3.2},
    'MLP 2 layers (Lab3)': {'R2_test': r2_test, 'RMSE': rmse_test}
}

```

Вывод сравнения

```

print("\n" + "="*70)

```

```

print("    СРАВНЕНИЕ МОДЕЛЕЙ РАЗЛИЧНОЙ СЛОЖНОСТИ")

```

```

print("="*70)

```

```

print(f"{'Модель':<25} {'R²':>12} {'RMSE, %':>12}")

```

```

print("-"*70)

```

```

for name, metrics in comparison.items():

```

```

    r2_val = metrics.get('R2_test') or metrics.get('R2_train')

```

```

    print(f"{'name':<25} {'r2_val':>12.3f} {'metrics['RMSE']:>12.2f}")

```

```

print("="*70)

```

Формирование сводки по текущей работе

```

results = {

```

```

    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),

```

```

    'student': '[Ваша фамилия]',

```

```

    'variant': '[Номер варианта]',

```

```

'num_features': num_features,
'architecture': f"[{units_layer1}, {units_layer2}, 1]",
'test_samples': len(y_test),
'R2_test': float(r2_test),
'RMSE_test': float(rmse_test),
'MAE_test': float(mae_test),
'prediction_params': [T_new, P_new, C_new] if num_features==3 else [T_new, P_new,
C_new, tau_new],
'prediction_yield': float(yield_pred)
}

# Вывод сводки
print("\n" + "="*60)

print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №3")
print("="*60)
for key, value in results.items():
    print(f"{key:25s}: {value}")
print("="*60)

# Сохранение модели и скалера
model_mlp.save('mlp_chem_model.keras')
print("    Модель сохранена: mlp_chem_model.keras")

# Опционально: сохранение scaler для будущего использования
# import joblib
# joblib.dump(scaler, 'scaler_chem.pkl')

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность разделения данных

1. Убедитесь, что `X_train_scaled.mean()` близко к 0, а `std()` близко к 1
2. Проверьте, что к тестовой выборке применяется только `transform()`, а не `fit_transform()`
3. Зафиксируйте размеры обучающей и тестовой выборок (должны быть ~80%/20%)

Тест 2: Работа EarlyStopping

1. Увеличьте `epochs=500` и уменьшите `patience=5` в Ячейке 6
2. Перезапустите обучение и проследите, на какой эпохе остановился процесс
3. Сравните графики обучения с разными настройками `patience`

Тест 3: Прогноз и интерпретация

1. В Ячейке 10 задайте экстремальные значения параметров (на границах диапазона)
2. Оцените, остается ли прогноз физически осмысленным
3. Прокомментируйте, как модель реагирует на изменение каждого фактора

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, количество признаков, архитектура MLP)
4. **Разработанные документы:**

— Скриншоты: динамика обучения (loss/MAE), графики «Факт vs Прогноз», анализ остатков

- Таблица метрик на тестовой выборке (MSE, RMSE, MAE, R^2)
- Сравнительная таблица моделей из лабораторных работ №1–3
- Результат прогноза для новых условий с физико-химической интерпретацией

5. **Выводы** (3–4 пункта):

- В чем преимущество MLP перед однослойной сетью для многофакторных задач?
- Почему важно оценивать модель на независимой тестовой выборке?
- Как параметр patience в EarlyStopping влияет на итоговое качество?
- Перспективы применения MLP для оптимизации реальных химических процессов

6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему стандартизацию признаков необходимо выполнять до разделения данных на обучающую и тестовую выборки, а `fit()` скалера применять только к обучающей части?

Вопрос 2:

Как функция активации ReLU в скрытых слоях позволяет нейронной сети моделировать сложные нелинейные зависимости, и почему на выходном слое используется linear?

Вопрос 3:

Что означает ситуация, когда `val_loss` начинает расти, а `train_loss` продолжает уменьшаться? Как EarlyStopping помогает предотвратить эту проблему?

Вопрос 4:

Как интерпретировать значение $RMSE = 2.5 \%$ в контексте химико-технологического процесса? В каких случаях такая точность может быть недостаточной?

Вопрос 5:

Какие дополнительные шаги необходимо предпринять, чтобы адаптировать данную программу для работы с реальными экспериментальными данными, содержащими пропуски, выбросы и категориальные признаки (тип катализатора, марка реактора)?

Лабораторная работа №4

«Исследование влияния функций активации на процесс обучения нейронной сети»

Цель: Экспериментально исследовать влияние различных функций активации (ReLU, sigmoid, tanh, linear) на скорость сходимости, стабильность обучения и итоговое качество нейронной сети при моделировании химико-технологических процессов.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 4: Функции активации

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №4
```

```
## Влияние функций активации на обучение нейронной сети
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
#### **Задание:**
```

Создать программу, которая:

1. Генерирует данные химического процесса с нелинейной зависимостью
2. Создает несколько идентичных архитектур нейросетей с разными функциями активации
3. Обучает модели и сравнивает динамику сходимости
4. Визуализирует влияние активации на качество прогноза
5. Формулирует рекомендации по выбору функции активации для химических задач

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
import pandas as pd
```

```

print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras: создание и обучение моделей")
print(" sklearn: предобработка и метрики")
print(" matplotlib: визуализация сравнения")

```

Ячейка 3: Генерация данных с нелинейной зависимостью

Создайте новую ячейку с кодом и скопируйте:

===== ГЕНЕРАЦИЯ ДАННЫХ ХИМИЧЕСКОГО ПРОЦЕССА =====

```

np.random.seed(42)
n_samples = 400

```

===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: задать параметры процесса и тип нелинейности

Вариант 1: сигмоидальная зависимость (насыщение при высоких T)

Вариант 2: экспоненциальный рост с последующим спадом

Вариант 3: комбинированная нелинейность с взаимодействием факторов

process_type = 'sigmoid_saturation' # Измените согласно варианту!

Диапазоны параметров

T_min, T_max = 20, 150 # Температура, °C

P_min, P_max = 1, 10 # Давление, атм

===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====

Генерация входных данных

T = np.random.uniform(T_min, T_max, n_samples)

P = np.random.uniform(P_min, P_max, n_samples)

Целевая функция в зависимости от типа процесса

if process_type == 'sigmoid_saturation':

Насыщение выхода при высокой температуре

Yield = 90 / (1 + np.exp(-0.05*(T - 80))) + 0.5*P + np.random.normal(0, 2.0, n_samples)

elif process_type == 'exponential_peak':

Пик выхода при оптимальной температуре

Yield = 80 * np.exp(-0.002*(T - 90)**2) + 0.3*P + np.random.normal(0, 2.0, n_samples)

else: # combined

Сложная зависимость с взаимодействием

Yield = (0.4*T + 0.6*P - 0.002*T**2 + 0.01*T*P +
np.random.normal(0, 2.0, n_samples))

Yield = np.clip(Yield, 0, 100)

Формирование матрицы признаков

X = np.column_stack([T, P])

y = Yield

print(f" Сгенерировано {n_samples} точек (тип: {process_type})")

```
print(f"    Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")
print(f"    Корреляция T-Y: {np.corrcoef(T, y)[0,1]:.3f}, P-Y: {np.corrcoef(P, y)[0,1]:.3f}")
```

Ячейка 4: Предобработка данных и разделение выборки

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ПОДГОТОВКА ДАННЫХ =====
```

```
# Стандартизация признаков
```

```
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

```
# Разделение на обучающую и тестовую выборки
```

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X_scaled, y, test_size=0.2, random_state=42
```

```
)
```

```
print(f"    Обучающая выборка: {X_train.shape[0]} образцов")
```

```
print(f"    Тестовая выборка: {X_test.shape[0]} образцов")
```

```
print(f"    Признаки стандартизированы: mean={X_train.mean():.3f}, std={X_train.std():.3f}")
```

Ячейка 5: Функция создания модели с заданной активацией (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 1: СОЗДАНИЕ МОДЕЛИ С ЗАДАННОЙ АКТИВАЦИЕЙ =====
```

```
def create_model_with_activation(input_dim, activation_hidden='relu', activation_output='linear',
    hidden_units=16):
```

```
    """
```

Создает нейронную сеть с указанной функцией активации в скрытых слоях.

Параметры:

- activation_hidden: 'relu', 'sigmoid', 'tanh', 'linear'

- activation_output: обычно 'linear' для регрессии

- hidden_units: количество нейронов в скрытом слое

```
    """
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

ЗАДАНИЕ: создать модель Sequential с двумя скрытыми слоями и указанной активацией

```
# Подсказка:
```

```
# - Первый Dense: hidden_units нейронов, activation=activation_hidden,
input_shape=(input_dim,)
```

```
# - Второй Dense: hidden_units//2 нейронов, activation=activation_hidden
```

```
# - Выходной Dense: 1 нейрон, activation=activation_output
```

```
# Ваш код ниже:
```

```
model = Sequential([
```

```
    Dense(hidden_units, activation=activation_hidden, input_shape=(input_dim,)),
```

```
    Dense(hidden_units//2, activation=activation_hidden),
```

```
    Dense(1, activation=activation_output)
```

)

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

return model

Тестирование функции

test_model = create_model_with_activation(input_dim=2, activation_hidden='relu')

print(" Функция создания модели работает")

test_model.summary()

Ячейка 6: Функция обучения и оценки модели (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 2: ФУНКЦИЯ ОБУЧЕНИЯ И ОЦЕНКИ =====

def train_and_evaluate(model, X_train, y_train, X_test, y_test, epochs=150, batch_size=16):

"""

Обучает модель и возвращает историю обучения и метрики на тесте.

"""

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: скомпилировать модель, обучить с EarlyStopping и оценить на тесте

Подсказка:

- compile: optimizer='adam', loss='mse', metrics=['mae']

- EarlyStopping: monitor='val_loss', patience=20, restore_best_weights=True

- fit: validation_split=0.2, callbacks=[early_stop], verbose=0

- predict и расчет метрик: MSE, RMSE, MAE, R²

Ваш код ниже:

Компиляция

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

Callback для ранней остановки

early_stop = EarlyStopping(
 monitor='val_loss',
 patience=20,
 restore_best_weights=True,
 verbose=0
)

Обучение

history = model.fit(
 X_train, y_train,
 epochs=epochs,
 batch_size=batch_size,
 validation_split=0.2,
 callbacks=[early_stop],
 verbose=0
)

Прогноз и метрики на тесте

```

y_pred = model.predict(X_test, verbose=0).flatten()
metrics = {
    'mse': mean_squared_error(y_test, y_pred),
    'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
    'mae': mean_absolute_error(y_test, y_pred),
    'r2': r2_score(y_test, y_pred),
    'epochs_trained': len(history.history['loss']),
    'final_train_loss': history.history['loss'][-1],
    'final_val_loss': history.history['val_loss'][-1]
}

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

return history, metrics

print("    Функция обучения и оценки готова к использованию")

Ячейка 7: Эксперимент: сравнение функций активации (ЗАДАНИЕ 3)
Создайте новую ячейку с кодом и скопируйте:
# ===== ЗАДАНИЕ 3: СРАВНЕНИЕ ФУНКЦИЙ АКТИВАЦИИ =====

# ===== ВАШ ВАРИАНТ: список активаций для сравнения =====
# Вариант 1: ['relu', 'sigmoid']
# Вариант 2: ['relu', 'tanh']
# Вариант 3: ['relu', 'sigmoid', 'tanh', 'linear']

activations_to_test = ['relu', 'sigmoid', 'tanh'] # Измените согласно варианту!

results = {}
histories = {}

print(f"    Запуск эксперимента: сравнение {len(activations_to_test)} функций активации")
print("-"*60)

for activation in activations_to_test:
    print(f"\n Обучение модели с активацией '{activation}'...")

    # Создание модели
    model = create_model_with_activation(
        input_dim=X_train.shape[1],
        activation_hidden=activation,
        hidden_units=16
    )

    # Обучение и оценка
    history, metrics = train_and_evaluate(model, X_train, y_train, X_test, y_test)

    # Сохранение результатов
    results[activation] = metrics
    histories[activation] = history

```

```
print(f"    Эпох: {metrics['epochs_trained']}, R²: {metrics['r2']:.3f}, RMSE: {metrics['rmse']:.2f}%")
```

```
print("\n" + "="*60)
```

```
print("    СВОДНАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ")
```

```
print("="*60)
```

```
# Формирование DataFrame для наглядного сравнения
```

```
results_df = pd.DataFrame(results).T
```

```
results_df = results_df[['r2', 'rmse', 'mae', 'epochs_trained', 'final_val_loss']]
```

```
results_df.columns = ['R²', 'RMSE', '%', 'MAE', '%', 'Эпох', 'Val Loss']
```

```
print(results_df.round(3))
```

```
print("="*60)
```

Ячейка 8: Визуализация динамики обучения (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 4: ГРАФИКИ ДИНАМИКИ ОБУЧЕНИЯ =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: построить сравнительные графики изменения потерь для разных активаций
```

```
# Ваш код ниже:
```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

```
# График 1: Валидационная потеря (log scale для наглядности)
```

```
for activation in activations_to_test:
```

```
    val_loss = histories[activation].history['val_loss']
```

```
    epochs = range(1, len(val_loss) + 1)
```

```
    axes[0].plot(epochs, val_loss, label=activation, linewidth=2)
```

```
axes[0].set_xlabel('Эпоха')
```

```
axes[0].set_ylabel('Валидационная потеря (MSE)')
```

```
axes[0].set_title('Сравнение сходимости: валидационная ошибка')
```

```
axes[0].legend()
```

```
axes[0].grid(True, alpha=0.3)
```

```
axes[0].set_yscale('log') # Логарифмическая шкала для лучшего визуального сравнения
```

```
# График 2: Финальные метрики (столбчатая диаграмма)
```

```
activations = list(results.keys())
```

```
r2_values = [results[a]['r2'] for a in activations]
```

```
rmse_values = [results[a]['rmse'] for a in activations]
```

```
x = np.arange(len(activations))
```

```
width = 0.35
```

```
axes[1].bar(x - width/2, r2_values, width, label='R²', color='skyblue', edgecolors='black')
```

```
axes[1].bar(x + width/2, rmse_values, width, label='RMSE', color='lightcoral',
```

```
edgecolors='black')
```

```
axes[1].set_xlabel('Функция активации')
```

```
axes[1].set_ylabel('Значение метрики')
```

```

axes[1].set_title('Финальное качество моделей на тестовой выборке')
axes[1].set_xticks(x)
axes[1].set_xticklabels(activations)
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

# Добавление значений на столбцы
for i, (r2, rmse) in enumerate(zip(r2_values, rmse_values)):
    axes[1].text(i - width/2, r2 + 0.02, f'{r2:.2f}', ha='center', fontsize=9)
    axes[1].text(i + width/2, rmse + 0.5, f'{rmse:.1f}', ha='center', fontsize=9)

plt.tight_layout()
plt.show()

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print("    Графики динамики обучения построены")

```

Ячейка 9: Анализ проблем обучения: затухающий градиент (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: АНАЛИЗ ПРОБЛЕМ ОБУЧЕНИЯ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: выявить признаки затухающего градиента и проанализировать стабильность обучения

Ваш код ниже:

```

print("\n    АНАЛИЗ СТАБИЛЬНОСТИ ОБУЧЕНИЯ")
print("="*60)

```

```

for activation in activations_to_test:
    metrics = results[activation]
    history = histories[activation]

```

```

train_loss = history.history['loss']
val_loss = history.history['val_loss']

```

```

# Признаки проблем обучения
issues = []

```

```

# 1. Затухающий градиент: потеря перестает уменьшаться рано
if metrics['epochs_trained'] < 50 and metrics['final_val_loss'] > 50:
    issues.append("    Возможное затухание градиента (ранняя остановка)")

```

```

# 2. Переобучение: вал. потеря растет при убывании обучающей
if len(val_loss) > 10:
    if val_loss[-1] > val_loss[len(val_loss)//2] * 1.5:
        issues.append("    Признаки переобучения (рост val_loss)")

```

```

# 3. Нестабильность: большие колебания потери

```

```

if np.std(val_loss[-10:]) > np.mean(val_loss[-10:]) * 0.3:
    issues.append("    Нестабильная сходимость (колебания val_loss)")

print(f"\n Активация: {activation}")
print(f"    R2 = {metrics['r2']:.3f}, RMSE = {metrics['rmse']:.2f}%")
print(f"    Эпох обучения: {metrics['epochs_trained']}")

if issues:
    for issue in issues:
        print(f"    {issue}")
    else:
        print("    Стабильное обучение без явных проблем")

print("\n" + "="*60)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 10: Прогноз для новых условий и сравнение активаций
Создайте новую ячейку с кодом и скопируйте:
# ===== ПРОГНОЗ ДЛЯ НОВЫХ УСЛОВИЙ =====

# ===== ВАШ ВАРИАНТ: параметры для прогноза =====
T_new = 95      # Температура, °C
P_new = 6.0     # Давление, атм

# Стандартизация новых данных
X_new = scaler.transform([[T_new, P_new]])

print(f"    Прогноз выхода продукта для условий: T={T_new}°C, P={P_new} атм")
print("-"*60)

predictions = {}

for activation in activations_to_test:
    # Создаем и загружаем веса лучшей модели (в реальном проекте – из файла)
    model = create_model_with_activation(
        input_dim=X_train.shape[1],
        activation_hidden=activation,
        hidden_units=16
    )
    # Быстрое дообучение на полных данных для прогноза
    model.compile(optimizer='adam', loss='mse')
    model.fit(X_train, y_train, epochs=10, verbose=0)

    # Прогноз
    pred = model.predict(X_new, verbose=0)[0, 0]
    predictions[activation] = pred

    print(f" {activation:10s} → {pred:6.2f} %")

print("-"*60)

```

```
# Визуализация разброса прогнозов
if len(predictions) > 1:
    pred_values = list(predictions.values())
    pred_names = list(predictions.keys())

    plt.figure(figsize=(8, 4))
    plt.barh(pred_names, pred_values, color='lightgreen', edgecolors='black')
    plt.xlabel('Прогнозируемый выход, %')
    plt.title('Разброс прогнозов в зависимости от функции активации')
    plt.grid(True, alpha=0.3, axis='x')
    plt.show()

    print(f"    Разброс прогнозов: {max(pred_values) - min(pred_values):.2f} %")
    print("    Большой разброс может указывать на неустойчивость модели")
```

Ячейка 11: Формирование отчета и сохранение результатов

Создайте новую ячейку с кодом и скопируйте:

===== ФОРМИРОВАНИЕ ОТЧЕТА И СОХРАНЕНИЕ =====

```
from datetime import datetime
```

```
# Выбор лучшей активации по критерию R²
best_activation = results_df['R²'].idxmax()
best_metrics = results[best_activation]
```

```
# Формирование сводки
```

```
report = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'process_type': process_type,
    'activations_tested': activations_to_test,
    'best_activation': best_activation,
    'best_R2': float(best_metrics['r2']),
    'best_RMSE': float(best_metrics['rmse']),
    'prediction_T': T_new,
    'prediction_P': P_new,
    'predictions_by_activation': {k: float(v) for k, v in predictions.items()}
}
```

```
# Вывод сводки
```

```
print("\n" + "="*70)
print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №4")
print("="*70)
for key, value in report.items():
    print(f"{key:30s}: {value}")
print("="*70)
```

```
# Рекомендации на основе результатов
```

```
print("\n    РЕКОМЕНДАЦИИ ПО ВЫБОРУ АКТИВАЦИИ:")
```

```

if best_activation == 'relu':
    print(" – ReLU показала лучшую сходимость – рекомендуется для большинства задач")
    print(" – Избегайте использования ReLU на выходном слое регрессии")
elif best_activation == 'tanh':
    print(" – Tanh обеспечил стабильное обучение – подходит для данных с
центрированным распределением")
    print(" – Требуется обязательной стандартизации входных признаков")
elif best_activation == 'sigmoid':
    print(" – Sigmoid показал приемлемые результаты – может использоваться для задач с
насыщением")
    print(" – Риск затухания градиента в глубоких сетях – ограничивайте количество
слоев")
else:
    print(" – Linear активация – только для выходного слоя регрессионных задач")

# Сохранение лучшей модели
best_model = create_model_with_activation(
    input_dim=X_train.shape[1],
    activation_hidden=best_activation,
    hidden_units=16
)
best_model.compile(optimizer='adam', loss='mse')
best_model.fit(X_train, y_train, epochs=best_metrics['epochs_trained'], verbose=0)
best_model.save(f'model_activation_{best_activation}.keras')
print(f"\n Лучшая модель сохранена: model_activation_{best_activation}.keras")

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность сравнения активаций

1. Убедитесь, что все модели обучаются без ошибок
2. Проверьте, что графики динамики обучения отображаются для всех активаций
3. Зафиксируйте, какая активация показала наименьшую валидационную потерю

Тест 2: Выявление проблем обучения

1. Добавьте в список `activations_to_test` функцию 'linear' для скрытых слоев
2. Наблюдайте, как изменится динамика обучения (ожидается медленная сходимость)
3. Проанализируйте вывод ячеек 9 на предмет выявления проблем

Тест 3: Интерпретация прогнозов

1. В Ячейке 10 задайте экстремальные значения параметров (на границах диапазона)
2. Сравните разброс прогнозов для разных активаций
3. Сформулируйте вывод: какая активация дает наиболее устойчивые прогнозы?

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, тип процесса, список тестируемых активаций)
4. **Разработанные документы:**
 - Скриншоты: сравнительные графики динамики обучения, столбчатая диаграмма метрик, визуализация разброса прогнозов
 - Таблица результатов для всех активаций (R^2 , RMSE, MAE, эпохи обучения)

- Выводы анализа стабильности обучения (признаки затухания градиента, переобучения)
 - Результат прогноза для новых условий с комментарием о влиянии активации
5. **Выводы** (3–4 пункта):
- Какая функция активации показала наилучшее качество и почему?
 - В каких случаях может возникнуть проблема затухающего градиента?
 - Почему для выходного слоя регрессии всегда используется linear?
 - Практические рекомендации по выбору активации для химических задач
6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему функция активации ReLU чаще всего рекомендуется для скрытых слоев нейронных сетей, и в каких случаях предпочтительнее использовать tanh или sigmoid?

Вопрос 2:

Что такое проблема затухающего градиента (vanishing gradient), и как выбор функции активации влияет на ее возникновение?

Вопрос 3:

Почему для задачи регрессии на выходном слое нейронной сети всегда используется линейная активация, независимо от активаций в скрытых слоях?

Вопрос 4:

Как по графику валидационной потери можно определить, что модель с данной функцией активации обучается нестабильно?

Вопрос 5:

Какие дополнительные эксперименты можно провести, чтобы более надежно выбрать функцию активации для конкретной задачи химико-технологического моделирования (например, кросс-валидация, тест на разных подвыборках)?

Лабораторная работа №5

«Процесс обучения нейронной сети: исследование влияния оптимизаторов, размера батча и количества эпох»

Цель: Экспериментально исследовать влияние гиперпараметров процесса обучения (оптимизатор, размер батча, количество эпох, скорость обучения) на скорость сходимости, стабильность и итоговое качество нейронной сети при моделировании химико-технологических процессов.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 5: Гиперпараметры обучения

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №5
```

```
## Гиперпараметры обучения нейронной сети
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует данные химического процесса с нелинейной зависимостью
2. Создает эталонную архитектуру нейронной сети
3. Проводит серию экспериментов с разными оптимизаторами, размерами батча и скоростями обучения
4. Визуализирует кривые обучения и сравнивает финальные метрики
5. Формулирует рекомендации по выбору рациональных гиперпараметров для химических задач

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import pandas as pd
import time
```

```
print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras: создание и обучение моделей")
print(" Оптимизаторы: SGD, Adam, RMSprop")
print(" sklearn: предобработка и метрики")
print(" matplotlib: визуализация экспериментов")
```

Ячейка 3: Генерация данных химического процесса

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ГЕНЕРАЦИЯ ДАННЫХ ХИМИЧЕСКОГО ПРОЦЕССА =====
```

```
np.random.seed(42)
n_samples = 500
```

```
# ===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: задать параметры процесса согласно варианту
```

```
# Вариант 1: умеренная нелинейность, 2 фактора
# Вариант 2: сильная нелинейность с взаимодействием, 3 фактора
# Вариант 3: зашумленные данные, 2 фактора
```

```
noise_level = 2.5 # Уровень шума: 1.5 (низкий), 2.5 (средний), 4.0 (высокий)
num_features = 2 # Количество входных признаков
```

```
# Диапазоны параметров
T_min, T_max = 25, 130 # Температура, °C
P_min, P_max = 1, 9 # Давление, атм
```

```
# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====
```

```
# Генерация входных данных
T = np.random.uniform(T_min, T_max, n_samples)
P = np.random.uniform(P_min, P_max, n_samples)
```

```
if num_features == 2:
    X = np.column_stack([T, P])
    # Целевая функция с нелинейностью
    Yield = (0.3*T + 0.9*P - 0.0018*T**2 + 0.02*T*P +
             np.random.normal(0, noise_level, n_samples))
else:
    C = np.random.uniform(0.2, 2.0, n_samples) # концентрация катализатора
    X = np.column_stack([T, P, C])
    Yield = (0.25*T + 0.7*P + 10*C - 0.0015*T**2 + 0.015*T*C +
             np.random.normal(0, noise_level, n_samples))
```

```
Yield = np.clip(Yield, 0, 100)
y = Yield
```

```
print(f"    Сгенерировано {n_samples} точек ({num_features} признаков,
шум={noise_level})")
print(f"    Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")
```

Ячейка 4: Предобработка данных и создание эталонной модели

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ПОДГОТОВКА ДАННЫХ И МОДЕЛИ =====
```

```
# Стандартизация признаков
```

```
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
```

```
# Разделение на обучающую и тестовую выборки
```

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X_scaled, y, test_size=0.2, random_state=42
```

```
)
```

```
print(f"    Обучающая выборка: {X_train.shape[0]}, Тестовая: {X_test.shape[0]}")
```

```
# Функция создания эталонной модели
```

```
def create_baseline_model(input_dim, hidden_units=20):
```

```
    model = Sequential([
```

```
        Dense(hidden_units, activation='relu', input_shape=(input_dim,)),
```

```
        Dense(hidden_units//2, activation='relu'),
```

```
        Dense(1, activation='linear')
```

```
    ])
```

```
    return model
```

```
# Тест создания модели
```

```
baseline = create_baseline_model(X_train.shape[1])
```

```
print("    Эталонная модель создана")
```

```
baseline.summary()
```

Ячейка 5: Функция обучения с фиксацией истории (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 1: ФУНКЦИЯ ОБУЧЕНИЯ С ФИКСАЦИЕЙ МЕТРИК
```

```
=====
```

```
def train_with_config(model, X_train, y_train, X_test, y_test,
                      optimizer='adam', epochs=100, batch_size=16,
                      learning_rate=0.001, use_early_stopping=True):
```

```
    """
```

```
    Обучает модель с заданными гиперпараметрами и возвращает историю и метрики.
```

```
    """
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: настроить оптимизатор, скомпилировать модель, обучить с
опциональным EarlyStopping
```

```
# Подсказка:
```

```
# - Для SGD: SGD(learning_rate=lr), для Adam: Adam(learning_rate=lr)
```

```
# - compile: loss='mse', metrics=['mae']
# - EarlyStopping: monitor='val_loss', patience=15, restore_best_weights=True
# - fit: validation_split=0.2, verbose=0
```

```
# Ваш код ниже:
```

```
# Выбор и настройка оптимизатора
if optimizer == 'sgd':
    opt = SGD(learning_rate=learning_rate)
elif optimizer == 'adam':
    opt = Adam(learning_rate=learning_rate)
elif optimizer == 'rmsprop':
    opt = RMSprop(learning_rate=learning_rate)
else:
    opt = Adam(learning_rate=learning_rate) # по умолчанию
```

```
# Компиляция модели
model.compile(optimizer=opt, loss='mse', metrics=['mae'])
```

```
# Callbacks
callbacks = []
if use_early_stopping:
    callbacks.append(EarlyStopping(
        monitor='val_loss',
        patience=15,
        restore_best_weights=True,
        verbose=0
    ))
```

```
# Обучение
history = model.fit(
    X_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_split=0.2,
    callbacks=callbacks,
    verbose=0
)
```

```
# Оценка на тесте
y_pred = model.predict(X_test, verbose=0).flatten()
metrics = {
    'r2': r2_score(y_test, y_pred),
    'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
    'mae': mean_absolute_error(y_test, y_pred),
    'epochs_trained': len(history.history['loss']),
    'final_train_loss': history.history['loss'][-1],
    'final_val_loss': history.history['val_loss'][-1]
}
```

```
# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====
```

```

return history, metrics

print("    Функция обучения с конфигурацией готова")

Ячейка 6: Эксперимент 1: Сравнение оптимизаторов (ЗАДАНИЕ 2)
Создайте новую ячейку с кодом и скопируйте:
# ===== ЗАДАНИЕ 2: СРАВНЕНИЕ ОПТИМИЗАТОРОВ =====

# ===== ВАШ ВАРИАНТ: список оптимизаторов для сравнения =====
# Вариант 1: ['adam', 'sgd']
# Вариант 2: ['adam', 'rmsprop']
# Вариант 3: ['adam', 'sgd', 'rmsprop']

optimizers_to_test = ['adam', 'sgd', 'rmsprop'] # Измените согласно варианту!

results_opt = {}
histories_opt = {}

print(f"    Эксперимент 1: сравнение оптимизаторов")
print("."*60)

for opt_name in optimizers_to_test:
    print(f"\n Оптимизатор: {opt_name.upper()}")

    # Создание свежей модели для каждого эксперимента
    model = create_baseline_model(X_train.shape[1])

    # Обучение с фиксированными параметрами
    history, metrics = train_with_config(
        model, X_train, y_train, X_test, y_test,
        optimizer=opt_name,
        epochs=150,
        batch_size=16,
        learning_rate=0.001,
        use_early_stopping=True
    )

    results_opt[opt_name] = metrics
    histories_opt[opt_name] = history

    print(f"    Эпох: {metrics['epochs_trained']}, R2: {metrics['r2']:.3f}, RMSE:
    {metrics['rmse']:.2f}%")

# Сводная таблица
print("\n" + "="*60)

print("    РЕЗУЛЬТАТЫ: ОПТИМИЗАТОРЫ")
print("="*60)
opt_df = pd.DataFrame(results_opt).T[['r2', 'rmse', 'mae', 'epochs_trained']]
opt_df.columns = ['R2', 'RMSE, %', 'MAE, %', 'Эпох']
print(opt_df.round(3))
print("="*60)

```

Ячейка 7: Эксперимент 2: Влияние размера батча (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 3: ВЛИЯНИЕ РАЗМЕРА БАТЧА =====
```

```
# ===== ВАШ ВАРИАНТ: значения batch_size для исследования =====
```

```
# Вариант 1: [8, 32]
```

```
# Вариант 2: [16, 64]
```

```
# Вариант 3: [8, 16, 32, 64]
```

```
batch_sizes_to_test = [8, 16, 32] # Измените согласно варианту!
```

```
results_batch = {}
```

```
histories_batch = {}
```

```
print(f"    Эксперимент 2: влияние размера батча")
```

```
print("-"*60)
```

```
for batch_size in batch_sizes_to_test:
```

```
    print(f"\n    Размер батча: {batch_size}")
```

```
    model = create_baseline_model(X_train.shape[1])
```

```
    history, metrics = train_with_config(
```

```
        model, X_train, y_train, X_test, y_test,
```

```
        optimizer='adam', # фиксируем лучший оптимизатор
```

```
        epochs=150,
```

```
        batch_size=batch_size,
```

```
        learning_rate=0.001,
```

```
        use_early_stopping=True
```

```
    )
```

```
    results_batch[batch_size] = metrics
```

```
    histories_batch[batch_size] = history
```

```
    print(f"        Эпох: {metrics['epochs_trained']}, R2: {metrics['r2']:.3f}, RMSE: {metrics['rmse']:.2f}%")
```

```
# Сводная таблица
```

```
print("\n" + "-"*60)
```

```
print("    РЕЗУЛЬТАТЫ: РАЗМЕР БАТЧА")
```

```
print("-"*60)
```

```
batch_df = pd.DataFrame(results_batch).T[['r2', 'rmse', 'mae', 'epochs_trained']]
```

```
batch_df.columns = ['R2', 'RMSE, %', 'MAE, %', 'Эпох']
```

```
print(batch_df.round(3))
```

```
print("-"*60)
```

Ячейка 8: Эксперимент 3: Влияние скорости обучения (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 4: ВЛИЯНИЕ СКОРОСТИ ОБУЧЕНИЯ =====
```

```

# ===== ВАШ ВАРИАНТ: значения learning_rate для исследования =====
# Вариант 1: [0.01, 0.001]
# Вариант 2: [0.001, 0.0001]
# Вариант 3: [0.01, 0.001, 0.0001]

learning_rates_to_test = [0.01, 0.001, 0.0001] # Измените согласно варианту!

results_lr = {}
histories_lr = {}

print(f"    Эксперимент 3: влияние скорости обучения")
print("-"*60)

for lr in learning_rates_to_test:
    print(f"\n Скорость обучения: {lr}")

    model = create_baseline_model(X_train.shape[1])

    history, metrics = train_with_config(
        model, X_train, y_train, X_test, y_test,
        optimizer='adam',
        epochs=150,
        batch_size=16,
        learning_rate=lr,
        use_early_stopping=True
    )

    results_lr[lr] = metrics
    histories_lr[lr] = history

    print(f"    Эпох: {metrics['epochs_trained']}, R2: {metrics['r2']:.3f}, RMSE:
{metrics['rmse']:.2f}%")

# Сводная таблица
print("\n" + "-"*60)
print("    РЕЗУЛЬТАТЫ: СКОРОСТЬ ОБУЧЕНИЯ")
print("-"*60)
lr_df = pd.DataFrame(results_lr).T[['r2', 'rmse', 'mae', 'epochs_trained']]
lr_df.columns = ['R2', 'RMSE', '%', 'MAE', '%', 'Эпох']
lr_df.index = lr_df.index.map(lambda x: f"{x:.0e}") # форматирование экспонентой
print(lr_df.round(3))
print("-"*60)

```

Ячейка 9: Визуализация кривых обучения (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: ВИЗУАЛИЗАЦИЯ КРИВЫХ ОБУЧЕНИЯ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: построить сравнительные графики динамики потерь для разных конфигураций

Ваш код ниже:

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# График 1: Оптимизаторы (валидационная потеря)
ax = axes[0, 0]
for opt_name in optimizers_to_test:
    val_loss = histories_opt[opt_name].history['val_loss']
    ax.plot(val_loss, label=opt_name.upper(), linewidth=2)
ax.set_xlabel('Эпоха')
ax.set_ylabel('Val Loss (MSE)')
ax.set_title('Сравнение оптимизаторов')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_yscale('log')

# График 2: Размер батча
ax = axes[0, 1]
for bs in batch_sizes_to_test:
    val_loss = histories_batch[bs].history['val_loss']
    ax.plot(val_loss, label=f'batch={bs}', linewidth=2)
ax.set_xlabel('Эпоха')
ax.set_ylabel('Val Loss (MSE)')
ax.set_title('Влияние размера батча')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_yscale('log')

# График 3: Скорость обучения
ax = axes[1, 0]
for lr in learning_rates_to_test:
    val_loss = histories_lr[lr].history['val_loss']
    ax.plot(val_loss, label=f'lr={lr:.0e}', linewidth=2)
ax.set_xlabel('Эпоха')
ax.set_ylabel('Val Loss (MSE)')
ax.set_title('Влияние скорости обучения')
ax.legend()
ax.grid(True, alpha=0.3)
ax.set_yscale('log')

# График 4: Сводная диаграмма финальных метрик
ax = axes[1, 1]
all_r2 = [results_opt['adam']['r2'],
          results_batch[16]['r2'],
          results_lr[0.001]['r2']]
labels = ['Adam (opt)', 'batch=16', 'lr=0.001']
colors = ['skyblue', 'lightgreen', 'lightcoral']

bars = ax.bar(labels, all_r2, color=colors, edgecolors='black')
ax.set_ylabel('R² на тестовой выборке')
ax.set_title('Финальное качество эталонных конфигураций')
ax.set_ylim(0, 1.1)
```

```

ax.grid(True, alpha=0.3, axis='y')

# Добавление значений на столбцы
for bar, val in zip(bars, all_r2):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.03,
            f'{val:.3f}', ha='center', fontsize=9)

plt.tight_layout()
plt.show()

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```
print("    Графики кривых обучения построены")
```

Ячейка 10: Анализ стабильности и выбор оптимальной конфигурации
 Создайте новую ячейку с кодом и скопируйте:
 # ===== АНАЛИЗ И ВЫБОР ОПТИМАЛЬНОЙ КОНФИГУРАЦИИ
 =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
 # ЗАДАНИЕ: проанализировать результаты и выбрать лучшую комбинацию гиперпараметров

Ваш код ниже:

```

print("\n    АНАЛИЗ СТАБИЛЬНОСТИ ОБУЧЕНИЯ")
print("="*70)

# Анализ оптимизаторов
print("\n    ОПТИМИЗАТОРЫ:")
best_opt = max(results_opt, key=lambda x: results_opt[x]['r2'])
for opt_name in optimizers_to_test:
    m = results_opt[opt_name]

    marker = "    " if opt_name == best_opt else " "
    print(f"    {marker} {opt_name:8s}: R²={m['r2']:.3f}, RMSE={m['rmse']:.2f}%,
    эпох={m['epochs_trained']}")

# Анализ размера батча
print("\n    РАЗМЕР БАТЧА:")
best_batch = max(results_batch, key=lambda x: results_batch[x]['r2'])
for bs in batch_sizes_to_test:
    m = results_batch[bs]

    marker = "    " if bs == best_batch else " "
    print(f"    {marker} batch={bs:2d}: R²={m['r2']:.3f}, RMSE={m['rmse']:.2f}%,
    эпох={m['epochs_trained']}")

# Анализ скорости обучения
print("\n    СКОРОСТЬ ОБУЧЕНИЯ:")
best_lr = max(results_lr, key=lambda x: results_lr[x]['r2'])
for lr in learning_rates_to_test:

```

```

m = results_lr[lr]
marker = "  " if lr == best_lr else "  "
print(f"  {marker} lr={lr:.0e}: R2={{m['r2']:.3f}}, RMSE={{m['rmse']:.2f}}%,
эпох={{m['epochs_trained']}}")

# Итоговая рекомендация
print("\n" + "="*70)

print("  РЕКОМЕНДУЕМАЯ КОНФИГУРАЦИЯ:")
print(f"  – Оптимизатор: {{best_opt.upper()}}")
print(f"  – Размер батча: {{best_batch}}")
print(f"  – Скорость обучения: {{best_lr:.0e}}")
print(f"  – Ожидаемое качество: R2 ≈ {{results_opt[best_opt]['r2']:.3f}}, RMSE ≈
{{results_opt[best_opt]['rmse']:.2f}}%")
print("="*70)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 11: Прогноз для новых условий и сохранение лучшей модели

Создайте новую ячейку с кодом и скопируйте:

===== ПРОГНОЗ И СОХРАНЕНИЕ МОДЕЛИ =====

===== ВАШ ВАРИАНТ: параметры для прогноза =====

T_new = 85 # Температура, °C

P_new = 5.5 # Давление, атм

Стандартизация новых данных

X_new = scaler.transform([[T_new, P_new]] if num_features==2 else [[T_new, P_new, 1.2]])

Создание и обучение финальной модели с лучшими гиперпараметрами

```

print(f"\n  Создание финальной модели с конфигурацией: {{best_opt}}, batch={{best_batch}},
lr={{best_lr:.0e}}")

```

```

final_model = create_baseline_model(X_train.shape[1])

```

```

history_final, metrics_final = train_with_config(

```

```

    final_model, X_train, y_train, X_test, y_test,

```

```

    optimizer=best_opt,

```

```

    epochs=200,

```

```

    batch_size=best_batch,

```

```

    learning_rate=best_lr,

```

```

    use_early_stopping=True

```

```

)

```

Прогноз

```

yield_pred = final_model.predict(X_new, verbose=0)[0, 0]

```

```

print(f"\n  Прогноз для новых условий: T={{T_new}}°C, P={{P_new}} атм")

```

```

print(f"  → Ожидаемый выход продукта: {{yield_pred:.2f}} %")

```

Проверка физической осмысленности

```

if 0 <= yield_pred <= 100:

```

```

    print("    Прогноз в физически допустимом диапазоне")
else:
    print("    Прогноз выходит за физические границы")

# Сохранение модели
final_model.save('final_optimized_model.keras')
print(f"\n    Финальная модель сохранена: final_optimized_model.keras")

```

Ячейка 12: Формирование сводного отчета

Создайте новую ячейку с кодом и скопируйте:

===== ФОРМИРОВАНИЕ СВОДНОГО ОТЧЕТА =====

```

from datetime import datetime

# Формирование сводки
report = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'noise_level': noise_level,
    'num_features': num_features,
    'optimizers_tested': optimizers_to_test,
    'batch_sizes_tested': batch_sizes_to_test,
    'learning_rates_tested': [f'{lr:.0e}' for lr in learning_rates_to_test],
    'best_optimizer': best_opt,
    'best_batch_size': best_batch,
    'best_learning_rate': f'{best_lr:.0e}',
    'final_R2_test': float(metrics_final['r2']),
    'final_RMSE_test': float(metrics_final['rmse']),
    'prediction_T': T_new,
    'prediction_P': P_new,
    'prediction_yield': float(yield_pred)
}

# Вывод сводки
print("\n" + "="*70)

print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №5")
print("="*70)
for key, value in report.items():
    print(f'{key:30s}: {value}')
print("="*70)

# Краткие выводы
print("\n    КЛЮЧЕВЫЕ ВЫВОДЫ:")
print("    1. Адаптивные оптимизаторы (Adam) обычно обеспечивают более быструю сходимость")
print("    2. Слишком большой batch_size может ухудшить обобщающую способность модели")
print("    3. Высокая learning_rate ускоряет обучение, но может привести к нестабильности")
print("    4. EarlyStopping критически важен для предотвращения переобучения")
print("    5. Оптимальные гиперпараметры зависят от конкретной задачи и данных")

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность экспериментов

1. Убедитесь, что все три эксперимента (оптимизаторы, батч, learning rate) выполняются без ошибок
2. Проверьте, что графики кривых обучения отображаются для всех конфигураций
3. Зафиксируйте, какая комбинация гиперпараметров показала наилучшее качество

Тест 2: Влияние экстремальных значений

1. Добавьте в `learning_rates_to_test` очень высокое значение (0.1) и очень низкое ($1e-6$)
2. Наблюдайте, как изменится динамика обучения (расхождение или крайне медленная сходимость)
3. Проанализируйте, почему такие значения не рекомендуются на практике

Тест 3: Интерпретация рекомендаций

1. Измените `noise_level` на более высокое значение (4.0) в Ячейке 3
2. Перезапустите эксперименты и сравните рекомендации по гиперпараметрам
3. Сформулируйте вывод: как уровень шума в данных влияет на выбор конфигурации обучения?

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, уровень шума, количество признаков, списки тестируемых гиперпараметров)
4. **Разработанные документы:**
 - Скриншоты: графики кривых обучения для оптимизаторов, размера батча и скорости обучения; сводная диаграмма финальных метрик
 - Три таблицы результатов экспериментов (с метриками R^2 , RMSE, MAE, эпохи обучения)
 - Блок анализа стабильности обучения с выделением лучшей конфигурации
 - Результат прогноза для новых условий с комментарием о физической осмысленности
5. **Выводы** (4–5 пунктов):
 - Какой оптимизатор показал наилучшее соотношение скорости сходимости и качества?
 - Как размер батча влияет на стабильность градиента и итоговую точность модели?
 - Почему выбор скорости обучения требует компромисса между скоростью и устойчивостью?
 - Как уровень шума в данных влияет на рекомендации по гиперпараметрам?
 - Практические рекомендации по настройке обучения для задач химико-технологического моделирования
6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

В чем принципиальное отличие адаптивных оптимизаторов (Adam, RMSprop) от классического SGD, и почему Adam чаще рекомендуется по умолчанию?

Вопрос 2:

Как размер батча влияет на оценку градиента, скорость обучения и обобщающую способность модели? В каких случаях предпочтительнее малый батч?

Вопрос 3:

Что происходит при слишком высокой и слишком низкой скорости обучения? Как по графику потерь диагностировать неудачный выбор learning rate?

Вопрос 4:

Почему при сравнении конфигураций гиперпараметров важно использовать фиксированную случайную инициализацию и одинаковые условия обучения?

Вопрос 5:

Какие дополнительные стратегии оптимизации гиперпараметров (например, случайный поиск, байесовская оптимизация) можно применить для автоматизации настройки модели в реальных проектах?

Лабораторная работа №6

«Предобработка данных для нейросетевых моделей в химических задачах»

Цель: Освоить базовые приемы подготовки разнородных технологических данных для обучения нейронных сетей: стандартизацию числовых признаков, кодирование категориальных переменных, корректное разделение выборки и обработку пропущенных значений, формируя культуру работы с реальными экспериментальными данными.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 6: Предобработка данных

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №6
```

```
## Предобработка данных для нейросетевого моделирования
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует смешанные данные химического процесса (числовые + категориальные признаки)
2. Выполняет очистку данных: обработка пропусков, выбросов
3. Кодировать категориальные переменные (One-Hot, Label Encoding)
4. Стандартизирует числовые признаки
5. Корректно разделяет данные на обучающую и тестовую выборки
6. Оценивает влияние предобработки на качество нейросетевой модели

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
from sklearn.model_selection import train_test_split
```

```

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder,
LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" sklearn: предобработка данных (Scaler, Encoder, Imputer)")
print(" pandas: работа с табличными данными")
print(" matplotlib/seaborn: визуализация распределений")

```

Ячейка 3: Генерация смешанных данных химического процесса

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ГЕНЕРАЦИЯ СМЕШАННЫХ ДАННЫХ =====

np.random.seed(42)
n_samples = 400

# ===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: задать параметры генерации данных согласно варианту

# Вариант 1: 2 числовых + 1 категориальный признак (2 класса)
# Вариант 2: 3 числовых + 1 категориальный признак (3 класса)
# Вариант 3: 2 числовых + 2 категориальных признака

num_numeric = 2          # Количество числовых признаков
num_categorical = 1       # Количество категориальных признаков
cat_classes = 2          # Количество классов в категориальном признаке
missing_ratio = 0.05      # Доля пропущенных значений (0.05 = 5%)

# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====

# Генерация числовых признаков
T = np.random.uniform(30, 130, n_samples)    # Температура, °C
P = np.random.uniform(1, 10, n_samples)       # Давление, атм
numeric_data = [T, P]

if num_numeric == 3:
    C = np.random.uniform(0.1, 2.5, n_samples) # Концентрация катализатора
    numeric_data.append(C)
    feature_names_num = ['T_C', 'P_atm', 'Catalyst_molL']
else:
    feature_names_num = ['T_C', 'P_atm']

# Генерация категориальных признаков
categorical_data = []
feature_names_cat = []

if num_categorical >= 1:
    catalyst_type = np.random.choice(['Type_A', 'Type_B', 'Type_C'][:cat_classes], n_samples)
    categorical_data.append(catalyst_type)

```

```

feature_names_cat.append('Catalyst_type')

if num_categorical >= 2:
    reactor_type = np.random.choice(['Batch', 'Continuous'], n_samples)
    categorical_data.append(reactor_type)
    feature_names_cat.append('Reactor_type')

# Формирование DataFrame
df = pd.DataFrame()
for name, data in zip(feature_names_num, numeric_data):
    df[name] = data
for name, data in zip(feature_names_cat, categorical_data):
    df[name] = data

# Целевая переменная с зависимостью от категориального признака
if num_categorical >= 1:
    cat_effect = np.where(df['Catalyst_type'] == 'Type_A', 5,
                          np.where(df['Catalyst_type'] == 'Type_B', 0, -3))
else:
    cat_effect = 0

Yield = (0.25*df['T_C'] + 0.8*df['P_atm'] + cat_effect +
         np.random.normal(0, 2.5, n_samples))
df['Yield_percent'] = np.clip(Yield, 0, 100)

# Добавление пропущенных значений (имитация реальных данных)
if missing_ratio > 0:
    for col in feature_names_num:
        missing_idx = np.random.choice(n_samples, size=int(n_samples * missing_ratio),
                                        replace=False)
        df.loc[missing_idx, col] = np.nan

print(f"   Сгенерировано {n_samples} записей")
print(f"   Признаки: числовые={feature_names_num},
категориальные={feature_names_cat}")
print(f"   Пропущенные значения: {df.isnull().sum().sum()}
({df.isnull().mean().mean()*100:.1f}%)")
print("\n   Первые 5 строк данных:")
print(df.head())

```

Ячейка 4: Разведочный анализ данных (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ЗАДАНИЕ 1: РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ =====

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: визуализировать распределения признаков и выявить аномалии

# Подсказка:
# - df.describe() для статистики числовых признаков
# - df['категория'].value_counts() для категориальных
# - sns.boxplot() для выявления выбросов

```

```

# - df.isnull().sum() для подсчета пропусков

# Ваш код ниже:

# Статистика числовых признаков
print("    Статистика числовых признаков:")
print(df[feature_names_num].describe())

# Распределение категориальных признаков
if feature_names_cat:
    print("\n    Распределение категориальных признаков:")
    for col in feature_names_cat:
        print(f"\n{col}:")
        print(df[col].value_counts())

# Визуализация выбросов
fig, axes = plt.subplots(1, len(feature_names_num), figsize=(5*len(feature_names_num), 4))
if len(feature_names_num) == 1:
    axes = [axes]

for ax, col in zip(axes, feature_names_num):
    sns.boxplot(y=df[col], ax=ax, color='lightblue')
    ax.set_title(f'{col}: выбросы')
    ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Тепловая карта пропусков
if df.isnull().sum().sum() > 0:
    plt.figure(figsize=(8, 3))
    sns.heatmap(df.isnull().T, cmap='viridis', cbar=False)
    plt.title('Карта пропущенных значений')
    plt.xlabel('Индекс образца')
    plt.ylabel('Признак')
    plt.show()

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print("    Разведочный анализ завершен")

```

Ячейка 5: Обработка пропущенных значений и выбросов (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 2: ОЧИСТКА ДАННЫХ =====

```

# Создаем копию для работы
df_clean = df.copy()

```

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: обработать пропуски и выбросы в числовых признаках

```

# Подсказка:
# - SimpleImputer(strategy='mean') для заполнения пропусков средним
# - Для выбросов: метод межквартильного размаха (IQR)
# Q1 = df.quantile(0.25), Q3 = df.quantile(0.75), IQR = Q3 - Q1
# Замена выбросов на границы: lower = Q1 - 1.5*IQR, upper = Q3 + 1.5*IQR

# Ваш код ниже:

# 1. Обработка пропусков: заполнение средним значением
imputer = SimpleImputer(strategy='mean')
df_clean[feature_names_num] = imputer.fit_transform(df_clean[feature_names_num])
print(f" Пропуски обработаны: осталось
{df_clean[feature_names_num].isnull().sum().sum()}")

# 2. Обработка выбросов методом IQR
for col in feature_names_num:
    Q1 = df_clean[col].quantile(0.25)
    Q3 = df_clean[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

# Подсчет выбросов до обработки
outliers_before = ((df_clean[col] < lower_bound) | (df_clean[col] > upper_bound)).sum()

# Замена выбросов на границы
df_clean[col] = np.clip(df_clean[col], lower_bound, upper_bound)

print(f" {col}: заменено {outliers_before} выбросов")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print("\n Очистка данных завершена")
print(" Проверка после обработки:")
print(df_clean[feature_names_num].describe())

```

Ячейка 6: Кодирование категориальных признаков (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 3: КОДИРОВАНИЕ КАТЕГОРИАЛЬНЫХ ПРИЗНАКОВ =====

```

df_encoded = df_clean.copy()

# ===== ВАШ ВАРИАНТ: метод кодирования =====
# Вариант 1: One-Hot Encoding для всех категориальных признаков
# Вариант 2: Label Encoding для признаков с упорядоченными классами
# Вариант 3: Комбинированный подход

encoding_method = 'onehot' # Измените: 'onehot' или 'label'

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

```

ЗАДАНИЕ: преобразовать категориальные признаки в числовой формат

Подсказка:

- One-Hot: pd.get_dummies(df, columns=[...], drop_first=True)

- Label Encoding: LabelEncoder().fit_transform(df[col])

- Важно: сохранить информацию о преобразовании для новых данных!

Ваш код ниже:

```
if encoding_method == 'onehot':
```

```
    # One-Hot Encoding с удалением первого столбца (избегаем мультиколлинеарности)
```

```
    df_encoded = pd.get_dummies(df_encoded, columns=feature_names_cat, drop_first=True)
```

```
    print("    Применен One-Hot Encoding (drop_first=True)")
```

```
else: # label encoding
```

```
    label_encoders = {}
```

```
    for col in feature_names_cat:
```

```
        le = LabelEncoder()
```

```
        df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
        label_encoders[col] = le
```

```
        print(f"    {col}: классы → {list(le.classes_)}")
```

```
    print("    Применен Label Encoding")
```

```
print(f"\n    Размерность после кодирования: {df_encoded.shape}")
```

```
print(f"    Признаки: {list(df_encoded.columns)}")
```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 7: Разделение выборки и стандартизация (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 4: РАЗДЕЛЕНИЕ И СТАНДАРТИЗАЦИЯ =====

Выделение признаков и целевой переменной

```
X = df_encoded.drop('Yield_percent', axis=1)
```

```
y = df_encoded['Yield_percent']
```

```
print(f"    Матрица признаков: {X.shape}, Целевая переменная: {y.shape}")
```

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: корректно разделить данные и выполнить стандартизацию

КРИТИЧЕСКИ ВАЖНО:

1. Сначала разделить на train/test

2. fit() скалера применять ТОЛЬКО к train

3. transform() применять и к train, и к test (но не fit_transform к test!)

Подсказка:

- train_test_split(X, y, test_size=0.2, random_state=42)

- StandardScaler().fit(X_train).transform(X_train)

- scaler.transform(X_test) для тестовой выборки

Ваш код ниже:

1. Разделение выборки

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
print(f"    Обучающая выборка: {X_train.shape[0]}, Тестовая: {X_test.shape[0]}")
```

2. Стандартизация числовых признаков

```
scaler = StandardScaler()
```

Применяем fit только к обучающей выборке!

```
X_train_scaled = X_train.copy()
```

```
X_test_scaled = X_test.copy()
```

```
X_train_scaled[feature_names_num] = scaler.fit_transform(X_train[feature_names_num])
```

```
X_test_scaled[feature_names_num] = scaler.transform(X_test[feature_names_num])
```

```
print("    Стандартизация выполнена")
```

```
print(f"    Проверка train: mean={X_train_scaled[feature_names_num].mean().mean():.3f},
std={X_train_scaled[feature_names_num].std().mean():.3f}")
```

```
print(f"    Проверка test: mean={X_test_scaled[feature_names_num].mean().mean():.3f},
std={X_test_scaled[feature_names_num].std().mean():.3f}")
```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 8: Обучение модели и оценка влияния предобработки (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: ОЦЕНКА ВЛИЯНИЯ ПРЕДОБРАБОТКИ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: обучить нейросеть на подготовленных данных и оценить качество

Подсказка:

- Создать простую модель: 2 скрытых слоя с ReLU, выход linear

- compile: optimizer='adam', loss='mse'

- fit: epochs=100, batch_size=16, validation_split=0.2, callbacks=[EarlyStopping]

- Оценить на тесте: R², RMSE, MAE

Ваш код ниже:

Создание модели

```
def create_simple_model(input_dim):
```

```
    return Sequential([
        Dense(16, activation='relu', input_shape=(input_dim,)),
        Dense(8, activation='relu'),
        Dense(1, activation='linear')
    ])
```

```
model = create_simple_model(X_train_scaled.shape[1])
```

```
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```

# Callback для ранней остановки
early_stop = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True,
verbose=0)

# Обучение
history = model.fit(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)

# Прогноз и метрики на тесте
y_pred = model.predict(X_test_scaled, verbose=0).flatten()
metrics = {
    'r2': r2_score(y_test, y_pred),
    'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
    'mae': mean_absolute_error(y_test, y_pred)
}

print("\n" + "="*60)
print("    КАЧЕСТВО МОДЕЛИ НА ПОДГОТОВЛЕННЫХ ДАННЫХ")
print("="*60)
print(f'{'Метрика':<20} {'Значение':>15}')
print("-"*60)
print(f'{'R²':<20} {metrics['r2']:>15.3f}')
print(f'{'RMSE, %':<20} {metrics['rmse']:>15.2f}')
print(f'{'MAE, %':<20} {metrics['mae']:>15.2f}')
print("="*60)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

# Визуализация прогнозов
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred, alpha=0.7, edgecolors='black')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Фактический выход, %')
plt.ylabel('Прогноз, %')
plt.title(f'Факт vs Прогноз (R² = {metrics["r2"]:.3f})')
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
residuals = y_test - y_pred
plt.scatter(y_pred, residuals, alpha=0.7, edgecolors='black')
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Прогноз, %')
plt.ylabel('Остаток')

```

```
plt.title('Анализ остатков')
plt.grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```

Ячейка 9: Эксперимент: сравнение методов предобработки (ЗАДАНИЕ 6)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 6: СРАВНЕНИЕ МЕТОДОВ ПРЕДОБРАБОТКИ
=====
```

```
# ===== ВАШ ВАРИАНТ: что сравнивать =====
```

```
# Вариант 1: StandardScaler vs MinMaxScaler
```

```
# Вариант 2: One-Hot vs Label Encoding
```

```
# Вариант 3: С обработкой выбросов и без
```

```
comparison_type = 'scaler' # Измените: 'scaler', 'encoding', 'outliers'
```

```
results_comparison = {}
```

```
print(f"    Эксперимент: сравнение методов ({comparison_type})")
print("-"*60)
```

```
if comparison_type == 'scaler':
```

```
    scalers_to_test = {'Standard': StandardScaler(), 'MinMax': MinMaxScaler()}
```

```
    for name, scaler_obj in scalers_to_test.items():
```

```
        # Повторяем подготовку с новым скалером
```

```
        X_train_s = X_train.copy()
```

```
        X_test_s = X_test.copy()
```

```
        X_train_s[feature_names_num] = scaler_obj.fit_transform(X_train[feature_names_num])
```

```
        X_test_s[feature_names_num] = scaler_obj.transform(X_test[feature_names_num])
```

```
        model = create_simple_model(X_train_s.shape[1])
```

```
        model.compile(optimizer='adam', loss='mse')
```

```
        model.fit(X_train_s, y_train, epochs=50, batch_size=16, validation_split=0.2, verbose=0)
```

```
        y_pred = model.predict(X_test_s, verbose=0).flatten()
```

```
        r2 = r2_score(y_test, y_pred)
```

```
        rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
        results_comparison[name] = {'R2': r2, 'RMSE': rmse}
```

```
        print(f"    {name:10s}: R²={r2:.3f}, RMSE={rmse:.2f}%")
```

```
elif comparison_type == 'encoding':
```

```
    # Сравнение One-Hot и Label (упрощенно)
```

```
    for enc_method in ['onehot', 'label']:
```

```
        df_temp = df_clean.copy()
```

```
        if enc_method == 'onehot':
```

```
            df_temp = pd.get_dummies(df_temp, columns=feature_names_cat, drop_first=True)
```

```
        else:
```

```
            for col in feature_names_cat:
```

```

df_temp[col] = LabelEncoder().fit_transform(df_temp[col])

X_temp = df_temp.drop('Yield_percent', axis=1)
y_temp = df_temp['Yield_percent']

X_tr, X_te, y_tr, y_te = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42)

scaler_temp = StandardScaler()
X_tr[feature_names_num] = scaler_temp.fit_transform(X_tr[feature_names_num])
X_te[feature_names_num] = scaler_temp.transform(X_te[feature_names_num])

model = create_simple_model(X_tr.shape[1])
model.compile(optimizer='adam', loss='mse')
model.fit(X_tr, y_tr, epochs=50, batch_size=16, verbose=0)

y_pred = model.predict(X_te, verbose=0).flatten()
r2 = r2_score(y_te, y_pred)
rmse = np.sqrt(mean_squared_error(y_te, y_pred))

results_comparison[enc_method] = {'R2': r2, 'RMSE': rmse}
print(f"  {enc_method:10s}: R2={r2:.3f}, RMSE={rmse:.2f}%")

# Визуализация сравнения
if results_comparison:
    plt.figure(figsize=(8, 5))
    names = list(results_comparison.keys())
    r2_vals = [results_comparison[n]['R2'] for n in names]

    plt.bar(names, r2_vals, color=['skyblue', 'lightcoral'][:len(names)], edgecolors='black')
    plt.ylabel('R² на тестовой выборке')
    plt.title(f'Сравнение методов: {comparison_type}')
    plt.ylim(0, 1.1)
    plt.grid(True, alpha=0.3, axis='y')

    for i, (name, r2) in enumerate(zip(names, r2_vals)):
        plt.text(i, r2 + 0.03, f'{r2:.3f}', ha='center')

    plt.show()

print("  Эксперимент завершен")

```

Ячейка 10: Прогноз для новых условий с применением предобработки

Создайте новую ячейку с кодом и скопируйте:

===== ПРОГНОЗ ДЛЯ НОВЫХ УСЛОВИЙ =====

===== ВАШ ВАРИАНТ: параметры для прогноза =====

T_new = 90 # Температура, °C

P_new = 5.0 # Давление, атм

catalyst_new = 'Type_A' # Тип катализатора (если есть категориальный признак)

print(f" Прогноз для условий: T={T_new}°C, P={P_new} атм", end="")

if 'Catalyst_type' in feature_names_cat:

```

    print(f", Catalyst={catalyst_new}")
else:
    print()

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: подготовить новые данные с применением той же предобработки

# КРИТИЧЕСКИ ВАЖНО:
# - Использовать тот же imputer, scaler, encoder, что и при обучении
# - Применять transform, а не fit_transform!

# Ваш код ниже:

# 1. Создание DataFrame с новыми данными
new_data = pd.DataFrame({
    'T_C': [T_new],
    'P_atm': [P_new]
})
if num_numeric == 3:
    new_data['Catalyst_molL'] = [1.2]

if 'Catalyst_type' in feature_names_cat:
    new_data['Catalyst_type'] = [catalyst_new]
if 'Reactor_type' in feature_names_cat:
    new_data['Reactor_type'] = ['Batch']

# 2. Применение той же предобработки
# Обработка пропусков (если есть)
new_data[feature_names_num] = imputer.transform(new_data[feature_names_num])

# Кодирование категориальных признаков
if encoding_method == 'onehot':
    new_data = pd.get_dummies(new_data, columns=feature_names_cat, drop_first=True)
    # Добавление отсутствующих столбцов с нулями
    for col in X.columns:
        if col not in new_data.columns:
            new_data[col] = 0
    new_data = new_data[X.columns] # Приведение к тому же порядку столбцов
else:
    for col in feature_names_cat:
        new_data[col] = label_encoders[col].transform(new_data[col])

# Стандартизация
new_data[feature_names_num] = scaler.transform(new_data[feature_names_num])

# 3. Прогноз
yield_pred = model.predict(new_data, verbose=0)[0, 0]

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print(f" → Ожидаемый выход продукта: {yield_pred:.2f} %")

```

```

if 0 <= yield_pred <= 100:
    print("    Прогноз в физически допустимом диапазоне")
else:
    print("    Прогноз выходит за физические границы – возможна экстраполяция")

```

Ячейка 11: Формирование отчета и сохранение артефактов

Создайте новую ячейку с кодом и скопируйте:

===== СОХРАНЕНИЕ РЕЗУЛЬТАТОВ И АРТЕФАКТОВ =====

```

from datetime import datetime
import joblib

# Формирование сводки
report = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'data_summary': {
        'samples': n_samples,
        'numeric_features': feature_names_num,
        'categorical_features': feature_names_cat,
        'missing_ratio': f"{missing_ratio*100:.1f}%"
    },
    'preprocessing': {
        'imputer': 'mean',
        'outlier_method': 'IQR',
        'encoding_method': encoding_method,
        'scaler': 'StandardScaler'
    },
    'model_metrics': {
        'R2_test': float(metrics['r2']),
        'RMSE_test': float(metrics['rmse']),
        'MAE_test': float(metrics['mae'])
    },
    'prediction': {
        'T_C': T_new,
        'P_atm': P_new,
        'Yield_pred': float(yield_pred)
    }
}

# Вывод сводки
print("\n" + "="*70)

print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №6")
print("="*70)
for key, value in report.items():
    if isinstance(value, dict):
        print(f"\n{key.upper()}:")
        for k, v in value.items():
            print(f"    {k:20s}: {v}")
    else:

```

```

    print(f'{key:30s}: {value}')
print("="*70)

# Сохранение артефактов для воспроизводимости
model.save('model_lab6.keras')
joblib.dump(scaler, 'scaler_lab6.pkl')
if encoding_method == 'label' and 'label_encoders' in locals():
    joblib.dump(label_encoders, 'encoders_lab6.pkl')
if 'imputer' in locals():
    joblib.dump(imputer, 'imputer_lab6.pkl')

print("\n    Сохранены артефакты:")
print("    – model_lab6.keras – обученная модель")
print("    – scaler_lab6.pkl – скалер для стандартизации")
if encoding_method == 'label':
    print("    – encoders_lab6.pkl – кодировщики категориальных признаков")
print("    – imputer_lab6.pkl – обработчик пропущенных значений")
print("\n    Для прогноза на новых данных необходимо загрузить все артефакты!")

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность предобработки

1. Убедитесь, что после стандартизации $\text{mean} \approx 0$ и $\text{std} \approx 1$ для обучающей выборки
2. Проверьте, что к тестовой выборке применяется только `transform()`, а не `fit_transform()`
3. Зафиксируйте количество обработанных пропусков и выбросов

Тест 2: Сравнение методов кодирования

1. Измените `encoding_method` на 'label' в Ячейке 6
2. Перезапустите обучение и сравните метрики с One-Hot Encoding
3. Проанализируйте, как метод кодирования влияет на качество при разном числе классов

Тест 3: Прогноз с применением артефактов

1. В Ячейке 10 задайте экстремальные значения параметров
2. Убедитесь, что новые данные проходят через тот же пайплайн предобработки
3. Оцените, насколько прогноз остается физически осмысленным

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, состав признаков, метод кодирования, тип скалера)
4. **Разработанные документы:**
 - Скриншоты: разведочный анализ (распределения, выбросы, карта пропусков), графики «Факт vs Прогноз», результаты сравнения методов предобработки
 - Таблицы: статистика до/после очистки, метрики качества модели, результаты эксперимента по сравнению методов
 - Блок с прогнозом для новых условий и проверкой физической осмысленности
5. **Выводы** (4–5 пунктов):

- Почему стандартизацию необходимо применять только к обучающей выборке (fit), а к тестовой – только преобразовывать (transform)?
- В каких случаях предпочтительнее One-Hot Encoding, а в каких – Label Encoding?
- Как обработка выбросов методом IQR влияет на качество модели?
- Почему сохранение артефактов предобработки критически важно для воспроизводимости?
- Практические рекомендации по подготовке реальных экспериментальных данных для нейросетевого моделирования

6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему применение `fit_transform()` к тестовой выборке является методологической ошибкой, и к каким последствиям это может привести?

Вопрос 2:

В чем разница между стандартизацией (`StandardScaler`) и нормализацией (`MinMaxScaler`)? В каких задачах химической технологии предпочтителен каждый из методов?

Вопрос 3:

Почему One-Hot Encoding с параметром `drop_first=True` помогает избежать проблемы мультиколлинеарности, и когда этим можно пренебречь?

Вопрос 4:

Как метод межквартильного размаха (IQR) для обработки выбросов соотносится с физико-химической природой экспериментальных данных? В каких случаях удаление выбросов может быть нежелательным?

Вопрос 5:

Какие дополнительные шаги необходимо предпринять при подготовке реальных лабораторных данных, содержащих несбалансированные классы, временные зависимости или иерархическую структуру (партии, серии экспериментов)?

Лабораторная работа №7

«Регуляризация и методы борьбы с переобучением в нейросетевом моделировании»

Цель: Освоить практические методы повышения обобщающей способности нейронных сетей (L2-регуляризация, Dropout, Early Stopping), научиться диагностировать переобучение по кривым обучения и выбирать рациональную стратегию регуляризации для задач моделирования химико-технологических процессов с ограниченным объемом экспериментальных данных.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 7: Регуляризация и борьба с переобучением

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №7
```

```
## Регуляризация и методы борьбы с переобучением
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует данные химического процесса с ограниченной выборкой и шумом
2. Создает нейронную сеть, склонную к переобучению (много параметров, мало данных)
3. Применяет методы регуляризации: L2, Dropout, Early Stopping
4. Визуализирует кривые обучения и диагностирует переобучение
5. Сравнивает качество моделей с регуляризацией и без нее на тестовой выборке
6. Формулирует рекомендации по выбору стратегии регуляризации для химических задач

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Dropout
```

```
from tensorflow.keras.regularizers import l1, l2, l1_l2
```

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
from sklearn.model_selection import train_test_split, cross_val_score, KFold
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
import pandas as pd
```

```
print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras: создание моделей с регуляризацией")
print(" Regularizers: L1, L2, Elastic Net")
print("🔗🔗 Callbacks: EarlyStopping, ReduceLROnPlateau")
print(" sklearn: кросс-валидация и метрики")
```

Ячейка 3: Генерация данных с ограниченной выборкой и шумом

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ГЕНЕРАЦИЯ ДАННЫХ С ОГРАНИЧЕННОЙ ВЫБОРКОЙ
=====
```

```
np.random.seed(42)
```

```
# ===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: задать параметры генерации согласно варианту
```

```
# Вариант 1: малая выборка (80 образцов), высокий шум
```

```
# Вариант 2: средняя выборка (150 образцов), умеренный шум
```

```
# Вариант 3: малая выборка (80 образцов), выбросы в данных
```

```
n_samples = 120          # Размер выборки: 80, 120, 150
```

```
noise_level = 3.5        # Уровень шума: 2.0 (низкий), 3.5 (средний), 6.0 (высокий)
```

```
add_outliers = False     # Добавлять ли выбросы: True/False
```

```
# Диапазоны параметров процесса
```

```
T_min, T_max = 30, 120   # Температура, °C
```

```
P_min, P_max = 1, 8      # Давление, атм
```

```
# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====
```

```
# Генерация входных данных
```

```
T = np.random.uniform(T_min, T_max, n_samples)
```

```
P = np.random.uniform(P_min, P_max, n_samples)
```

```
# Целевая функция с нелинейностью
```

```
Yield = (0.3*T + 0.9*P - 0.002*T**2 + 0.015*T*P +
          np.random.normal(0, noise_level, n_samples))
```

```
# Добавление выбросов (если задано)
```

```
if add_outliers and n_samples >= 10:
```

```
    outlier_idx = np.random.choice(n_samples, size=max(3, n_samples//20), replace=False)
```

```
    Yield[outlier_idx] += np.random.choice([-1, 1], size=len(outlier_idx)) *
```

```
    np.random.uniform(10, 20, size=len(outlier_idx))
```

```
Yield = np.clip(Yield, 0, 100)
```

```
# Формирование матрицы признаков
```

```
X = np.column_stack([T, P])
```

```

y = Yield

# Стандартизация
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Разделение на обучающую и тестовую выборки (малая обучающая часть для имитации
дефицита данных)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.25, random_state=42
)

print(f"    Сгенерировано {n_samples} точек (шум={noise_level}, выбросы={add_outliers})")
print(f"    Обучающая выборка: {X_train.shape[0]}, Тестовая: {X_test.shape[0]}")
print(f"    Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")

```

Ячейка 4: Создание модели без регуляризации – демонстрация переобучения (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 1: МОДЕЛЬ БЕЗ РЕГУЛЯРИЗАЦИИ =====

```

def create_model_no_reg(input_dim, hidden_units=32):
    """
    Создает модель, склонную к переобучению: много параметров, нет регуляризации.
    """
    model = Sequential([
        Dense(hidden_units, activation='relu', input_shape=(input_dim,)),
        Dense(hidden_units, activation='relu'),
        Dense(hidden_units//2, activation='relu'),
        Dense(1, activation='linear')
    ])
    return model

# Создание и обучение модели без регуляризации
model_no_reg = create_model_no_reg(X_train.shape[1])
model_no_reg.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Обучение без EarlyStopping – даем модели "разогнаться"
history_no_reg = model_no_reg.fit(
    X_train, y_train,
    epochs=200,
    batch_size=8, # малый батч для усиления шума градиента
    validation_split=0.2,
    verbose=0
)

# Оценка на обучающей и тестовой выборках
y_train_pred_no_reg = model_no_reg.predict(X_train, verbose=0).flatten()
y_test_pred_no_reg = model_no_reg.predict(X_test, verbose=0).flatten()

r2_train_no_reg = r2_score(y_train, y_train_pred_no_reg)

```

```

r2_test_no_reg = r2_score(y_test, y_test_pred_no_reg)
rmse_test_no_reg = np.sqrt(mean_squared_error(y_test, y_test_pred_no_reg))

print("    Модель БЕЗ регуляризации:")
print(f"    R² train: {r2_train_no_reg:.3f}, R² test: {r2_test_no_reg:.3f}")
print(f"    Разница  $\Delta R^2$ : {r2_train_no_reg - r2_test_no_reg:+.3f} {'    переобучение' if r2_train_no_reg - r2_test_no_reg > 0.15 else '    ОК'}")
print(f"    RMSE test: {rmse_test_no_reg:.2f}%")

```

Ячейка 5: Визуализация переобучения (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 2: ДИАГНОСТИКА ПЕРЕОБУЧЕНИЯ =====
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: построить графики, позволяющие визуально диагностировать переобучение
```

```
# Подсказка:
```

```
# - График 1: train_loss vs val_loss – расхождение указывает на переобучение
```

```
# - График 2: предсказания на train и test – чрезмерная точность на train при плохом test
```

```
# - График 3: веса модели – большие значения весов могут указывать на неустойчивость
```

```
# Ваш код ниже:
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
```

```
# График 1: Кривые обучения
```

```

axes[0].plot(history_no_reg.history['loss'], label='Train Loss', linewidth=2)
axes[0].plot(history_no_reg.history['val_loss'], label='Val Loss', linewidth=2)
axes[0].set_xlabel('Эпоха')
axes[0].set_ylabel('MSE (потеря)')
axes[0].set_title('Кривые обучения: признак переобучения')
axes[0].legend()
axes[0].grid(True, alpha=0.3)
axes[0].set_yscale('log')

```

```
# График 2: Факт vs Прогноз (train и test)
```

```

axes[1].scatter(y_train, y_train_pred_no_reg, alpha=0.5, label='Train', color='blue', s=20)
axes[1].scatter(y_test, y_test_pred_no_reg, alpha=0.8, label='Test', color='red', s=30,
edgecolors='black')
axes[1].plot([y.min(), y.max()], [y.min(), y.max()], 'k--', linewidth=1)
axes[1].set_xlabel('Фактический выход, %')
axes[1].set_ylabel('Прогноз, %')
axes[1].set_title('Факт против Прогноза')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

```

```
# График 3: Распределение весов модели (индикатор устойчивости)
```

```

weights = []
for layer in model_no_reg.layers:
    if isinstance(layer, Dense):
        w = layer.get_weights()[0]

```

```

        weights.extend(w.flatten())
axes[2].hist(np.abs(weights), bins=30, color='lightcoral', edgecolors='black')
axes[2].set_xlabel('Абсолютное значение веса')
axes[2].set_ylabel('Частота')
axes[2].set_title('Распределение весов модели')
axes[2].grid(True, alpha=0.3)

```

```

plt.tight_layout()
plt.show()

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

print("    Графики диагностики переобучения построены")

```

```

print("    Признаки переобучения: вал. потеря растёт, а обучающая падает; большие значения весов")

```

Ячейка 6: Применение L2-регуляризации (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 3: L2-РЕГУЛЯРИЗАЦИЯ =====

===== ВАШ ВАРИАНТ: коэффициент регуляризации =====

Вариант 1: l2_reg = 0.001

Вариант 2: l2_reg = 0.01

Вариант 3: l2_reg = 0.1

l2_reg = 0.01 # Измените согласно варианту!

```

def create_model_l2(input_dim, hidden_units=32, l2_lambda=0.01):

```

```

    """

```

```

    Создает модель с L2-регуляризацией на всех скрытых слоях.

```

```

    """

```

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: добавить kernel_regularizer=l2(l2_lambda) к слоям Dense

Подсказка:

- from tensorflow.keras.regularizers import l2

- Dense(..., kernel_regularizer=l2(l2_lambda))

Ваш код ниже:

```

model = Sequential([

```

```

    Dense(hidden_units, activation='relu', input_shape=(input_dim,),

```

```

        kernel_regularizer=l2(l2_lambda)),

```

```

    Dense(hidden_units, activation='relu', kernel_regularizer=l2(l2_lambda)),

```

```

    Dense(hidden_units//2, activation='relu', kernel_regularizer=l2(l2_lambda)),

```

```

    Dense(1, activation='linear')

```

```

])

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

return model

```

```

# Создание и обучение модели с L2
model_l2 = create_model_l2(X_train.shape[1], l2_lambda=l2_reg)
model_l2.compile(optimizer='adam', loss='mse', metrics=['mae'])

history_l2 = model_l2.fit(
    X_train, y_train,
    epochs=200,
    batch_size=8,
    validation_split=0.2,
    verbose=0
)

# Оценка
y_test_pred_l2 = model_l2.predict(X_test, verbose=0).flatten()
r2_test_l2 = r2_score(y_test, y_test_pred_l2)
rmse_test_l2 = np.sqrt(mean_squared_error(y_test, y_test_pred_l2))

print(f"    Модель с L2-регуляризацией ( $\lambda=\{l2\_reg\}$ ):")
print(f"    R2 test: {r2_test_l2:.3f}, RMSE test: {rmse_test_l2:.2f}%")
print(f"    Улучшение  $\Delta$ RMSE: {rmse_test_no_reg - rmse_test_l2:+.2f}%")

```

Ячейка 7: Применение Dropout (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 4: DROPOUT =====

===== ВАШ ВАРИАНТ: коэффициент Dropout =====

Вариант 1: dropout_rate = 0.2

Вариант 2: dropout_rate = 0.4

Вариант 3: dropout_rate = 0.6

dropout_rate = 0.3 # Измените согласно варианту!

```

def create_model_dropout(input_dim, hidden_units=32, dropout=0.3):
    """

```

Создает модель с Dropout-слоями после каждого скрытого слоя.
 """

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: добавить слой Dropout(rate=dropout) после каждого Dense

Подсказка:

- from tensorflow.keras.layers import Dropout

- Последовательность: Dense → Dropout → Dense → Dropout → ...

Ваш код ниже:

```

model = Sequential([
    Dense(hidden_units, activation='relu', input_shape=(input_dim,)),
    Dropout(dropout),
    Dense(hidden_units, activation='relu'),
    Dropout(dropout),
    Dense(hidden_units//2, activation='relu'),
    Dropout(dropout),

```

```

        Dense(1, activation='linear')
    ])

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

    return model

# Создание и обучение модели с Dropout
model_dropout = create_model_dropout(X_train.shape[1], dropout=dropout_rate)
model_dropout.compile(optimizer='adam', loss='mse', metrics=['mae'])

history_dropout = model_dropout.fit(
    X_train, y_train,
    epochs=200,
    batch_size=8,
    validation_split=0.2,
    verbose=0
)

# Оценка
y_test_pred_dropout = model_dropout.predict(X_test, verbose=0).flatten()
r2_test_dropout = r2_score(y_test, y_test_pred_dropout)
rmse_test_dropout = np.sqrt(mean_squared_error(y_test, y_test_pred_dropout))

print(f"    Модель с Dropout (rate={dropout_rate}):")
print(f"    R2 test: {r2_test_dropout:.3f}, RMSE test: {rmse_test_dropout:.2f}%")
print(f"    Улучшение ΔRMSE: {rmse_test_no_reg - rmse_test_dropout:+.2f}%")

```

Ячейка 8: Применение Early Stopping (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 5: EARLY STOPPING =====
```

```
# ===== ВАШ ВАРИАНТ: параметр patience =====
```

```
# Вариант 1: patience = 10
```

```
# Вариант 2: patience = 20
```

```
# Вариант 3: patience = 30
```

```
patience = 15 # Измените согласно варианту!
```

```
# Создание модели (без регуляризации, но с EarlyStopping)
```

```
model_early = create_model_no_reg(X_train.shape[1])
```

```
model_early.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: настроить callback EarlyStopping и обучить модель
```

```
# Подсказка:
```

```
# - EarlyStopping(monitor='val_loss', patience=patience, restore_best_weights=True)
```

```
# - callbacks=[early_stop] в model.fit()
```

```
# Ваш код ниже:
```

```

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=patience,
    restore_best_weights=True, # Восстановить веса лучшей эпохи
    verbose=0
)

history_early = model_early.fit(
    X_train, y_train,
    epochs=300, # Большое число эпох – EarlyStopping остановит вовремя
    batch_size=8,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Оценка

```

y_test_pred_early = model_early.predict(X_test, verbose=0).flatten()
r2_test_early = r2_score(y_test, y_test_pred_early)
rmse_test_early = np.sqrt(mean_squared_error(y_test, y_test_pred_early))
epochs_trained = len(history_early.history['loss'])

```

```

print(f" Модель с EarlyStopping (patience={patience}):")
print(f" Остановлено на эпохе: {epochs_trained} из 300")
print(f" R2 test: {r2_test_early:.3f}, RMSE test: {rmse_test_early:.2f}%")
print(f" Улучшение ΔRMSE: {rmse_test_no_reg - rmse_test_early:+.2f}%")

```

Ячейка 9: Сравнительная визуализация методов регуляризации (ЗАДАНИЕ 6)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 6: СРАВНЕНИЕ МЕТОДОВ РЕГУЛЯРИЗАЦИИ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: построить сводные графики для сравнения эффективности методов

Ваш код ниже:

Подготовка данных для сравнения

```

methods = ['Без регул.', 'L2', 'Dropout', 'EarlyStop']
r2_test_values = [r2_test_no_reg, r2_test_l2, r2_test_dropout, r2_test_early]
rmse_test_values = [rmse_test_no_reg, rmse_test_l2, rmse_test_dropout, rmse_test_early]
colors = ['lightgray', 'skyblue', 'lightgreen', 'lightcoral']

```

```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
```

График 1: Столбчатая диаграмма R²

```

bars1 = axes[0].bar(methods, r2_test_values, color=colors, edgecolors='black')
axes[0].set_ylabel('R2 на тестовой выборке')
axes[0].set_title('Сравнение качества моделей')
axes[0].set_ylim(0, 1.1)

```

```

axes[0].grid(True, alpha=0.3, axis='y')
for bar, val in zip(bars1, r2_test_values):
    axes[0].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.03,
        f'{val:.3f}', ha='center', fontsize=9)

# График 2: Столбчатая диаграмма RMSE
bars2 = axes[1].bar(methods, rmse_test_values, color=colors, edgecolors='black')
axes[1].set_ylabel('RMSE, %')
axes[1].set_title('Сравнение ошибки прогноза')
axes[1].grid(True, alpha=0.3, axis='y')
for bar, val in zip(bars2, rmse_test_values):
    axes[1].text(bar.get_x() + bar.get_width()/2, val + 0.3,
        f'{val:.1f}', ha='center', fontsize=9)

plt.tight_layout()
plt.show()

# Сводная таблица
print("\n" + "="*70)

print("    СВОДНАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ")
print("="*70)
print(f'{"Метод":<15} {"R² test":>12} {"RMSE test, %":>15} {"Улучшение ΔRMSE":>18}')
print("-"*70)
for i, method in enumerate(methods):
    improvement = rmse_test_no_reg - rmse_test_values[i]
    print(f'{method:<15} {r2_test_values[i]:>12.3f} {rmse_test_values[i]:>15.2f} {improvement:>18.2f}')
print("-"*70)

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

# Выделение лучшего метода
best_idx = np.argmax(r2_test_values)
print(f"\n    Лучший метод по критерию R²: {methods[best_idx]}")

```

Ячейка 10: Комбинированная регуляризация и финальный прогноз

Создайте новую ячейку с кодом и скопируйте:

```

# ===== КОМБИНИРОВАННАЯ РЕГУЛЯРИЗАЦИЯ И ПРОГНОЗ
=====

```

```

# ===== ВАШ ВАРИАНТ: комбинация методов =====

```

```

# Вариант 1: L2 + EarlyStopping

```

```

# Вариант 2: Dropout + EarlyStopping

```

```

# Вариант 3: L2 + Dropout + EarlyStopping

```

```

combo_strategy = 'all' # Измените: 'l2_early', 'dropout_early', 'all'

```

```

print(f"    Создание модели с комбинированной регуляризацией: {combo_strategy}")

```

```

def create_model_combined(input_dim, hidden_units=24, l2_lambda=0.01, dropout=0.3):
    model = Sequential()

```

```

        model.add(Dense(hidden_units, activation='relu', input_shape=(input_dim,),
                        kernel_regularizer=l2(l2_lambda) if 'l2' in combo_strategy or combo_strategy ==
'all' else None))
        if 'dropout' in combo_strategy or combo_strategy == 'all':
            model.add(Dropout(dropout))
        else:
            model.add(Dropout(0)) # нейтральный слой

        model.add(Dense(hidden_units//2, activation='relu',
                        kernel_regularizer=l2(l2_lambda) if 'l2' in combo_strategy or combo_strategy ==
'all' else None))
        if 'dropout' in combo_strategy or combo_strategy == 'all':
            model.add(Dropout(dropout))

        model.add(Dense(1, activation='linear'))
        return model

# Создание и обучение финальной модели
model_final = create_model_combined(X_train.shape[1])
model_final.compile(optimizer='adam', loss='mse', metrics=['mae'])

early_stop_final = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True,
verbose=0)

history_final = model_final.fit(
    X_train, y_train,
    epochs=300,
    batch_size=8,
    validation_split=0.2,
    callbacks=[early_stop_final],
    verbose=0
)

# Прогноз для новых условий
# ===== ВАШ ВАРИАНТ: параметры для прогноза =====
T_new = 85      # Температура, °C
P_new = 5.0     # Давление, атм

X_new = scaler.transform([[T_new, P_new]])
yield_pred_final = model_final.predict(X_new, verbose=0)[0, 0]

# Оценка на тесте
y_test_pred_final = model_final.predict(X_test, verbose=0).flatten()
r2_test_final = r2_score(y_test, y_test_pred_final)
rmse_test_final = np.sqrt(mean_squared_error(y_test, y_test_pred_final))

print(f"\n    Прогноз для условий: T={T_new}°C, P={P_new} атм")
print(f"    → Ожидаемый выход продукта: {yield_pred_final:.2f} %")

print(f"\n    Финальная модель ({combo_strategy}):")
print(f"    R² test: {r2_test_final:.3f}, RMSE test: {rmse_test_final:.2f} %")

```

```
print(f" Улучшение относительно базовой:  $\Delta RMSE = \{rmse\_test\_no\_reg - rmse\_test\_final:+.2f\}\%$ ")
```

```
if 0 <= yield_pred_final <= 100:
```

```
    print("    Прогноз в физически допустимом диапазоне")
else:
```

```
    print("    Прогноз выходит за физические границы")
```

```
# Сохранение модели
```

```
model_final.save('model_regularized.keras')
```

```
print(f"\n    Финальная модель сохранена: model_regularized.keras")
```

Ячейка 11: Формирование сводного отчета

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ФОРМИРОВАНИЕ СВОДНОГО ОТЧЕТА =====
```

```
from datetime import datetime
```

```
# Формирование сводки
```

```
report = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'data_config': {
        'n_samples': n_samples,
        'noise_level': noise_level,
        'add_outliers': add_outliers
    },
    'regularization_params': {
        'l2_lambda': l2_reg if 'l2_reg' in locals() else None,
        'dropout_rate': dropout_rate if 'dropout_rate' in locals() else None,
        'early_stop_patience': patience if 'patience' in locals() else None,
        'combo_strategy': combo_strategy
    },
    'results_comparison': {
        'no_reg': {'R2': float(r2_test_no_reg), 'RMSE': float(rmse_test_no_reg)},
        'l2': {'R2': float(r2_test_l2), 'RMSE': float(rmse_test_l2)},
        'dropout': {'R2': float(r2_test_dropout), 'RMSE': float(rmse_test_dropout)},
        'early_stop': {'R2': float(r2_test_early), 'RMSE': float(rmse_test_early)},
        'combined': {'R2': float(r2_test_final), 'RMSE': float(rmse_test_final)}
    },
    'prediction': {
        'T_C': T_new,
        'P_atm': P_new,
        'Yield_pred': float(yield_pred_final)
    }
}
```

```
# Вывод сводки
```

```
print("\n" + "="*70)
```

```
print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №7")
```

```

print("="*70)
for key, value in report.items():
    if isinstance(value, dict):
        print(f"\n{key.upper().replace('_', ' ')}:")
        for k, v in value.items():
            print(f"  {k:25s}: {v}")
    else:
        print(f"{key:30s}: {value}")
print("="*70)

```

Краткие выводы

```

print("\n  КЛЮЧЕВЫЕ ВЫВОДЫ:")
print("  1. Переобучение диагностируется по расхождению train/val loss и большим весам")
print("  2. L2-регуляризация ограничивает рост весов, повышая устойчивость модели")
print("  3. Dropout эффективно снижает ко-адаптацию нейронов, особенно при малых выборках")
print("  4. EarlyStopping – простой и надежный способ предотвратить переобучение")
print("  5. Комбинация методов дает наилучший результат, но требует тонкой настройки")

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Диагностика переобучения

1. Убедитесь, что в модели без регуляризации R^2_{train} значительно превышает R^2_{test} (разница > 0.15)
2. Проверьте, что на графике кривых обучения валидационная потеря начинает расти после некоторой эпохи
3. Зафиксируйте значения весов модели – большие абсолютные значения могут указывать на неустойчивость

Тест 2: Сравнение методов регуляризации

1. Измените параметры регуляризации в соответствии с вариантом (λ , dropout_rate, patience)
2. Перезапустите обучение и сравните метрики для всех методов
3. Определите, какой метод дал наибольшее улучшение $\Delta RMSE$ относительно базовой модели

Тест 3: Комбинированная регуляризация

1. В Ячейке 10 измените combo_strategy на другую комбинацию
2. Оцените, дает ли комбинация методов синергетический эффект или ухудшает качество
3. Сформулируйте рекомендацию: при каких условиях (объем данных, уровень шума) какая стратегия предпочтительна

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, параметры генерации данных, значения гиперпараметров регуляризации)
4. **Разработанные документы:**
 - Скриншоты: графики диагностики переобучения (кривые обучения, факт/прогноз, распределение весов), сравнительные диаграммы метрик для всех методов

- Сводная таблица результатов (метод, R^2 test, RMSE test, Δ RMSE относительно базовой модели)
- Блок с прогнозом для новых условий и проверкой физической осмысленности

5. **Выводы** (4–5 пунктов):

- По каким признакам можно диагностировать переобучение нейронной сети?
- Как L2-регуляризация влияет на значения весов и обобщающую способность модели?
- Почему Dropout эффективен именно во время обучения, а не инференса?
- Как параметр patience в EarlyStopping влияет на итоговое качество и время обучения?
- Практические рекомендации по выбору стратегии регуляризации для задач химической технологии с ограниченным объемом экспериментальных данных

6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Что такое переобучение (overfitting) и как его можно диагностировать по кривым обучения и распределению весов модели?

Вопрос 2:

В чем заключается механизм действия L2-регуляризации, и как коэффициент λ влияет на компромисс между точностью на обучающей выборке и обобщающей способностью?

Вопрос 3:

Почему Dropout применяется только во время обучения, и как он помогает снизить ко-адаптацию нейронов в скрытых слоях?

Вопрос 4:

Как параметр patience в EarlyStopping влияет на процесс обучения? Что произойдет, если задать слишком малое или слишком большое значение?

Вопрос 5:

В каких ситуациях химико-технологического моделирования (малый объем данных, высокий шум, наличие выбросов) какие методы регуляризации будут наиболее эффективны, и почему?

Лабораторная работа №8

«Сравнительный анализ архитектур нейронных сетей для задач химической технологии»

Цель: Сформировать компетенции системного сравнения различных архитектур нейронных сетей (разная глубина, ширина, тип соединений), научиться оценивать компромисс между точностью прогноза, вычислительной сложностью и интерпретируемостью модели, вырабатывать обоснованные рекомендации по выбору архитектуры для конкретных задач химико-технологического моделирования.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 8: Сравнение архитектур нейросетей

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

```
# Лабораторная работа №8
```

```
## Сравнительный анализ архитектур нейронных сетей
```

```
**Студент:** [Ваша фамилия и имя]
```

```
**Дата:** [Сегодняшняя дата]
```

```
**Вариант:** [Номер варианта]
```

```
### **Задание:**
```

Создать программу, которая:

1. Генерирует данные химического процесса со сложной нелинейной зависимостью
2. Реализует функцию построения модели с конфигурируемой архитектурой
3. Обучает и сравнивает 3–4 архитектуры нейронных сетей (разная глубина/ширина)
4. Оценивает компромисс между точностью, сложностью и временем обучения
5. Визуализирует результаты сравнения и формулирует рекомендации по выбору архитектуры
6. Выполняет прогноз с использованием лучшей модели

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential, Model
```

```
from tensorflow.keras.layers import Dense, Dropout, Input
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import time
```

```
print(" Библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras Functional API: создание сложных архитектур")
print(" time: измерение времени обучения")
print(" sklearn: метрики качества и предобработка")
```

Ячейка 3: Генерация данных со сложной нелинейной зависимостью

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ГЕНЕРАЦИЯ ДАННЫХ СО СЛОЖНОЙ ЗАВИСИМОСТЬЮ
=====
```

```
np.random.seed(42)
n_samples = 600
```

```
# ===== ВАШ ВАРИАНТ НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: задать параметры процесса и тип зависимости согласно варианту
```

```
# Вариант 1: 3 фактора, полиномиальная зависимость 3-й степени
```

```
# Вариант 2: 4 фактора, экспоненциально-тригонометрическая зависимость
```

```
# Вариант 3: 3 фактора, кусочно-нелинейная зависимость с пороговыми эффектами
```

```
num_features = 3          # Количество входных признаков
complexity_type = 'poly3'  # Тип сложности: 'poly3', 'exp_trig', 'piecewise'
noise_level = 2.0          # Уровень шума
```

```
# Диапазоны параметров
```

```
T_min, T_max = 25, 140    # Температура, °C
```

```
P_min, P_max = 1, 10      # Давление, атм
```

```
C_min, C_max = 0.1, 3.0   # Концентрация катализатора, моль/л
```

```
# ===== ВАШ ВАРИАНТ ЗАКОНЧИЛСЯ =====
```

```
# Генерация входных данных
```

```
T = np.random.uniform(T_min, T_max, n_samples)
```

```
P = np.random.uniform(P_min, P_max, n_samples)
```

```
C = np.random.uniform(C_min, C_max, n_samples)
```

```
# Целевая функция в зависимости от типа сложности
```

```
if complexity_type == 'poly3':
```

```
    # Полином 3-й степени с взаимодействиями
```

```
    Yield = (0.2*T + 0.7*P + 8*C -
              0.001*T**2 + 0.00002*T**3 +
              0.01*T*P + 0.05*P*C - 0.003*T*C +
              np.random.normal(0, noise_level, n_samples))
```

```
elif complexity_type == 'exp_trig':
```

```
    # Экспоненциально-тригонометрическая зависимость
```

```
    Yield = (50 * (1 - np.exp(-0.02*(T-30))) * np.cos(0.03*P) +
```

```

        5*np.sin(2*C) + 0.3*T*P/100 +
        np.random.normal(0, noise_level, n_samples))
else: # piecewise
    # Кусочно-нелинейная с пороговыми эффектами
    Yield = np.where(T < 70,
        0.3*T + 0.5*P + 5*C,
        0.15*T + 1.2*P + 12*C - 0.002*T**2)
    Yield += np.random.normal(0, noise_level, n_samples)

Yield = np.clip(Yield, 0, 100)

# Формирование матрицы признаков
X = np.column_stack([T, P, C])
feature_names = ['T_C', 'P_atm', 'Catalyst_molL']
y = Yield

# Стандартизация и разделение
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

print(f"    Сгенерировано {n_samples} точек ({complexity_type}, {num_features} признака)")
print(f"    Обучающая: {X_train.shape[0]}, Тестовая: {X_test.shape[0]}")
print(f"    Диапазон выхода: {y.min():.1f} – {y.max():.1f} %")

```

Ячейка 4: Функция создания модели с конфигурируемой архитектурой (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 1: ФУНКЦИЯ СОЗДАНИЯ МОДЕЛИ С
ПАРАМЕТРИЗОВАННОЙ АРХИТЕКТУРОЙ =====

```

def create_architecture_model(input_dim, architecture_config, dropout_rate=0.0,
    output_activation='linear'):
    """

```

Создает нейронную сеть по заданной конфигурации.

Параметры:

- architecture_config: список, задающий количество нейронов в каждом скрытом слое
Пример: [16] → 1 слой с 16 нейронами; [32, 16, 8] → 3 слоя
- dropout_rate: коэффициент Dropout (0.0 = без регуляризации)
- output_activation: функция активации выходного слоя

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: реализовать построение модели Sequential по конфигурации

Подсказка:

- # - Первый слой: input_shape=(input_dim,)
- # - Цикл по architecture_config для добавления скрытых слоев
- # - Опционально: Dropout после каждого скрытого слоя
- # - Выходной слой: Dense(1, activation=output_activation)

```

# Ваш код ниже:

model = Sequential()

# Первый скрытый слой (с указанием input_shape)
if len(architecture_config) > 0:
    model.add(Dense(architecture_config[0], activation='relu', input_shape=(input_dim,)))
    if dropout_rate > 0:
        model.add(Dropout(dropout_rate))

# Остальные скрытые слои
for units in architecture_config[1:]:
    model.add(Dense(units, activation='relu'))
    if dropout_rate > 0:
        model.add(Dropout(dropout_rate))

# Выходной слой
model.add(Dense(1, activation=output_activation))

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

return model

# Тестирование функции
test_configs = {
    'shallow_wide': [32],
    'deep_narrow': [8, 8, 8, 8],
    'balanced': [16, 8]
}

print("    Тест создания моделей:")
for name, config in test_configs.items():
    model = create_architecture_model(X_train.shape[1], config)
    param_count = model.count_params()
    print(f"    {name:15s}: архитектура {config} → {param_count} параметров")

```

Ячейка 5: Функция обучения и оценки модели (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 2: ФУНКЦИЯ ОБУЧЕНИЯ И КОМПЛЕКСНОЙ ОЦЕНКИ =====

```

def train_and_evaluate_architecture(model, X_train, y_train, X_test, y_test,
                                     epochs=150, batch_size=16, model_name='Model'):
    """
    Обучает модель и возвращает комплексные метрики: качество, время, сложность.
    """
    # ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
    # ЗАДАНИЕ: реализовать обучение, оценку и сбор метрик

    # Подсказка:
    # - Замер времени: time.time() до и после fit()

```

```

# - EarlyStopping: monitor='val_loss', patience=20, restore_best_weights=True
# - Метрики: R2, RMSE, MAE на тесте + количество параметров + время обучения
# - Возврат: словарь с метриками и история обучения

# Ваш код ниже:

# Компиляция
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Callback
early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True,
verbose=0)

# Замер времени обучения
start_time = time.time()

# Обучение
history = model.fit(
    X_train, y_train,
    epochs=epochs,
    batch_size=batch_size,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)

train_time = time.time() - start_time

# Прогноз и метрики на тесте
y_pred = model.predict(X_test, verbose=0).flatten()

metrics = {
    'name': model_name,
    'architecture': [layer.units for layer in model.layers if isinstance(layer, Dense) and
layer.units != 1],
    'n_parameters': model.count_params(),
    'train_time_sec': train_time,
    'epochs_trained': len(history.history['loss']),
    'r2_test': r2_score(y_test, y_pred),
    'rmse_test': np.sqrt(mean_squared_error(y_test, y_pred)),
    'mae_test': mean_absolute_error(y_test, y_pred),
    'final_train_loss': history.history['loss'][-1],
    'final_val_loss': history.history['val_loss'][-1]
}

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

return history, metrics

print("    Функция обучения и оценки готова")

```

Ячейка 6: Определение архитектур для сравнения (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 3: ОПРЕДЕЛЕНИЕ АРХИТЕКТУР ДЛЯ СРАВНЕНИЯ
=====
```

```
# ===== ВАШ ВАРИАНТ: набор архитектур для тестирования =====
```

```
# Вариант 1: 3 архитектуры (shallow, deep, balanced)
```

```
# Вариант 2: 4 архитектуры (+ very_deep или very_wide)
```

```
# Вариант 3: 3 архитектуры с разными dropout_rate
```

```
n_architectures = 3 # Измените согласно варианту!
```

```
# Базовые конфигурации
```

```
architectures = {
    'shallow_wide': {
        'config': [64],
        'dropout': 0.0,
        'description': '1 слой × 64 нейрона (широкая)'
    },
    'deep_narrow': {
        'config': [16, 16, 16, 16],
        'dropout': 0.0,
        'description': '4 слоя × 16 нейронов (глубокая)'
    },
    'balanced': {
        'config': [32, 16],
        'dropout': 0.0,
        'description': '2 слоя × [32, 16] (сбалансированная)'
    }
}
```

```
# Добавление четвертой архитектуры для Варианта 2
```

```
if n_architectures == 4:
    architectures['very_deep'] = {
        'config': [12, 12, 12, 12, 12],
        'dropout': 0.1,
        'description': '5 слоев × 12 нейронов + Dropout (очень глубокая)'
    }
```

```
# Добавление Dropout для Варианта 3
```

```
if n_architectures == 3 and 'variant3' in locals():
    for key in architectures:
        architectures[key]['dropout'] = 0.2
```

```
print(f"    Планируется сравнение {len(architectures)} архитектур:")
```

```
for name, arch in architectures.items():
```

```
    print(f"    - {name:15s}: {arch['description']}")
```

Ячейка 7: Запуск сравнительного эксперимента (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 4: ЗАПУСК СРАВНИТЕЛЬНОГО ЭКСПЕРИМЕНТА
=====
```

```

results = {}
histories = {}

print("\n" + "="*70)
print("    ЗАПУСК СРАВНИТЕЛЬНОГО ЭКСПЕРИМЕНТА")
print("="*70)

for arch_name, arch_params in architectures.items():
    print(f"\n Архитектура: {arch_name} – {arch_params['description']}")

    # Создание модели
    model = create_architecture_model(
        input_dim=X_train.shape[1],
        architecture_config=arch_params['config'],
        dropout_rate=arch_params['dropout']
    )

    # Обучение и оценка
    history, metrics = train_and_evaluate_architecture(
        model, X_train, y_train, X_test, y_test,
        epochs=150, batch_size=16, model_name=arch_name
    )

    # Сохранение результатов
    results[arch_name] = metrics
    histories[arch_name] = history

    # Вывод ключевых метрик
    print(f"    Параметры: {metrics['n_parameters']:,}, Эпох: {metrics['epochs_trained']},
    Время: {metrics['train_time_sec']:.1f} с")
    print(f"    R² = {metrics['r2_test']:.3f}, RMSE = {metrics['rmse_test']:.2f}%, MAE =
    {metrics['mae_test']:.2f}%")

    # Сводная таблица
    print("\n" + "="*70)
    print("    СВОДНАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ")
    print("="*70)

    results_df = pd.DataFrame(results).T
    results_df = results_df[['r2_test', 'rmse_test', 'mae_test', 'n_parameters', 'train_time_sec',
    'epochs_trained']]
    results_df.columns = ['R²', 'RMSE, %', 'MAE, %', 'Параметры', 'Время, с', 'Эпох']
    results_df['Параметры'] = results_df['Параметры'].apply(lambda x: f'{x:,}')
    results_df['Время, с'] = results_df['Время, с'].apply(lambda x: f'{x:.1f}')

    print(results_df.round(3))
    print("="*70)

```

Ячейка 8: Визуализация сравнения архитектур (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: ВИЗУАЛИЗАЦИЯ СРАВНЕНИЯ =====

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: построить комплексные графики для сравнения архитектур
```

```
# Ваш код ниже:
```

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
# График 1:  $R^2$  vs Количество параметров (компромисс точность/сложность)
```

```
ax = axes[0, 0]
```

```
for arch_name in results:
```

```
    m = results[arch_name]
```

```
    ax.scatter(m['n_parameters'], m['r2_test'], s=100, label=arch_name,  
              edgecolors='black', alpha=0.8)
```

```
    ax.annotate(arch_name, (m['n_parameters'], m['r2_test']),  
               textcoords="offset points", xytext=(5,5), fontsize=8)
```

```
ax.set_xlabel('Количество параметров модели')
```

```
ax.set_ylabel('R2 на тестовой выборке')
```

```
ax.set_title('Точность vs Сложность модели')
```

```
ax.grid(True, alpha=0.3)
```

```
ax.set_xscale('log')
```

```
# График 2: RMSE vs Время обучения (компромисс точность/скорость)
```

```
ax = axes[0, 1]
```

```
for arch_name in results:
```

```
    m = results[arch_name]
```

```
    ax.scatter(m['train_time_sec'], m['rmse_test'], s=100, label=arch_name,  
              edgecolors='black', alpha=0.8)
```

```
    ax.annotate(arch_name, (m['train_time_sec'], m['rmse_test']),  
               textcoords="offset points", xytext=(5,5), fontsize=8)
```

```
ax.set_xlabel('Время обучения, с')
```

```
ax.set_ylabel('RMSE, %')
```

```
ax.set_title('Точность vs Вычислительные затраты')
```

```
ax.grid(True, alpha=0.3)
```

```
# График 3: Кривые обучения (валидационная потеря)
```

```
ax = axes[1, 0]
```

```
for arch_name in architectures:
```

```
    val_loss = histories[arch_name].history['val_loss']
```

```
    ax.plot(val_loss, label=arch_name, linewidth=2)
```

```
ax.set_xlabel('Эпоха')
```

```
ax.set_ylabel('Val Loss (MSE)')
```

```
ax.set_title('Сравнение сходимости обучения')
```

```
ax.legend(fontsize=8)
```

```
ax.grid(True, alpha=0.3)
```

```
ax.set_yscale('log')
```

```
# График 4: Столбчатая диаграмма финальных метрик
```

```
ax = axes[1, 1]
```

```
arch_names = list(results.keys())
```

```
r2_vals = [results[a]['r2_test'] for a in arch_names]
```

```
rmse_vals = [results[a]['rmse_test'] for a in arch_names]
```

```

x = np.arange(len(arch_names))
width = 0.35

bars1 = ax.bar(x - width/2, r2_vals, width, label='R²', color='skyblue', edgecolors='black')
bars2 = ax.bar(x + width/2, rmse_vals, width, label='RMSE', color='lightcoral',
edgecolors='black')

ax.set_xlabel('Архитектура')
ax.set_ylabel('Значение метрики')
ax.set_title('Финальное качество на тестовой выборке')
ax.set_xticks(x)
ax.set_xticklabels([a.replace('_', '\n') for a in arch_names], fontsize=9)
ax.legend()
ax.grid(True, alpha=0.3, axis='y')

# Добавление значений на столбцы
for bar, val in zip(bars1, r2_vals):
    ax.text(bar.get_x() + bar.get_width()/2, val + 0.02, f'{val:.2f}', ha='center', fontsize=8)
for bar, val in zip(bars2, rmse_vals):
    ax.text(bar.get_x() + bar.get_width()/2, val + 0.5, f'{val:.1f}', ha='center', fontsize=8)

plt.tight_layout()
plt.show()

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

print("    Графики сравнения архитектур построены")

```

Ячейка 9: Анализ компромиссов и выбор оптимальной архитектуры (ЗАДАНИЕ 6)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 6: АНАЛИЗ КОМПРОМИССОВ И ВЫБОР ЛУЧШЕЙ
АРХИТЕКТУРЫ =====

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: проанализировать результаты и выбрать архитектуру по заданному критерию

Подсказка:

- Критерии выбора: максимальный R^2 , минимальный RMSE, лучший баланс
точность/сложность

- Баланс можно оценить как: $\text{score} = R^2 / \log(\text{параметры})$ или $R^2 / \text{время_обучения}$

Ваш код ниже:

```

print("\n    АНАЛИЗ КОМПРОМИССОВ И ВЫБОР ОПТИМАЛЬНОЙ АРХИТЕКТУРЫ")
print("="*70)

```

Выбор критерия оптимизации

===== ВАШ ВАРИАНТ: критерий выбора =====

Вариант 1: по максимальному R^2

Вариант 2: по минимальному RMSE

```

# Вариант 3: по балансу:  $R^2$  / log(параметры)

selection_criterion = 'balanced' # Измените: 'r2_max', 'rmse_min', 'balanced'

if selection_criterion == 'r2_max':
    best_arch = max(results, key=lambda x: results[x]['r2_test'])
    criterion_name = "Максимальный  $R^2$ "
elif selection_criterion == 'rmse_min':
    best_arch = min(results, key=lambda x: results[x]['rmse_test'])
    criterion_name = "Минимальный RMSE"
else: # balanced
    best_arch = max(results, key=lambda x: results[x]['r2_test'] /
np.log1p(results[x]['n_parameters']))
    criterion_name = "Баланс  $R^2$  / log(параметры)"

best_metrics = results[best_arch]

print(f" Критерий выбора: {criterion_name}")
print(f" Выбранная архитектура: {best_arch}")
print(f" Описание: {architectures[best_arch]['description']}")
print(f" Параметры: {best_metrics['n_parameters']:,}, Время:
{best_metrics['train_time_sec']:.1f}c")
print(f" Качество:  $R^2$  = {best_metrics['r2_test']:.3f}, RMSE =
{best_metrics['rmse_test']:.2f}%")

# Сравнение с другими архитектурами
print(f"\n Относительное улучшение относительно других архитектур:")
for arch_name in results:
    if arch_name != best_arch:
        m = results[arch_name]
        delta_r2 = best_metrics['r2_test'] - m['r2_test']
        delta_rmse = m['rmse_test'] - best_metrics['rmse_test']
        print(f" vs {arch_name:15s}:  $\Delta R^2$  = {delta_r2:+.3f},  $\Delta RMSE$  = {delta_rmse:+.2f}%")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 10: Прогноз с лучшей моделью и интерпретация результатов

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ПРОГНОЗ С ЛУЧШЕЙ МОДЕЛЬЮ И ИНТЕРПРЕТАЦИЯ
=====

```

```

# ===== ВАШ ВАРИАНТ: параметры для прогноза =====

```

```

T_new = 95          # Температура, °C
P_new = 6.0         # Давление, атм
C_new = 1.8         # Концентрация катализатора, моль/л

```

```

print(f"\n Прогноз с моделью '{best_arch}' для условий:")
print(f" T = {T_new} °C, P = {P_new} атм, C = {C_new} моль/л")

```

```

# Подготовка новых данных

```

```

X_new = scaler.transform([[T_new, P_new, C_new]])

```

```

# Прогноз
best_model = create_architecture_model(
    input_dim=X_train.shape[1],
    architecture_config=architectures[best_arch]['config'],
    dropout_rate=architectures[best_arch]['dropout']
)
best_model.compile(optimizer='adam', loss='mse')
# Быстрое дообучение на полных данных для прогноза
best_model.fit(X_train, y_train, epochs=best_metrics['epochs_trained'], verbose=0)

yield_pred = best_model.predict(X_new, verbose=0)[0, 0]

print(f" → Ожидаемый выход продукта: {yield_pred:.2f} %")

# Интерпретация в контексте химической технологии
print(f"\n ИНТЕРПРЕТАЦИЯ РЕЗУЛЬТАТА:")
if yield_pred > 80:
    print(" – Высокий выход продукта – режим близок к оптимальному")
    print(" – Рекомендуется валидация на пилотной установке")
elif yield_pred > 60:
    print(" – Умеренный выход – возможен потенциал для оптимизации")
    print(" – Целесообразно исследовать окрестность параметров")
else:
    print(" – Низкий выход – режим требует корректировки")
    print(" – Рассмотреть изменение соотношения параметров")

# Проверка физической осмысленности
if 0 <= yield_pred <= 100:
    print(" Прогноз в физически допустимом диапазоне")
else:
    print(" Прогноз выходит за физические границы – возможна экстраполяция")

# Сохранение лучшей модели
best_model.save(f'model_best_{best_arch}.keras')
print(f"\n Лучшая модель сохранена: model_best_{best_arch}.keras")

```

Ячейка 11: Формирование сводного отчета

Создайте новую ячейку с кодом и скопируйте:

===== ФОРМИРОВАНИЕ СВОДНОГО ОТЧЕТА =====

```

from datetime import datetime

# Формирование сводки
report = {
    'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
    'student': '[Ваша фамилия]',
    'variant': '[Номер варианта]',
    'data_config': {
        'n_samples': n_samples,

```

```

        'complexity_type': complexity_type,
        'num_features': num_features
    },
    'architectures_tested': {name: arch['description'] for name, arch in architectures.items()},
    'selection_criterion': selection_criterion,
    'best_architecture': {
        'name': best_arch,
        'description': architectures[best_arch]['description'],
        'config': architectures[best_arch]['config'],
        'dropout': architectures[best_arch]['dropout']
    },
    'best_metrics': {
        'R2_test': float(best_metrics['r2_test']),
        'RMSE_test': float(best_metrics['rmse_test']),
        'MAE_test': float(best_metrics['mae_test']),
        'n_parameters': best_metrics['n_parameters'],
        'train_time_sec': float(best_metrics['train_time_sec'])
    },
    'prediction': {
        'T_C': T_new,
        'P_atm': P_new,
        'Catalyst_molL': C_new,
        'Yield_pred': float(yield_pred)
    }
}

```

Вывод сводки

```
print("\n" + "="*70)
```

```
print("    СВОДКА ПО ЛАБОРАТОРНОЙ РАБОТЕ №8")
```

```
print("="*70)
```

```
for key, value in report.items():
```

```
    if isinstance(value, dict):
```

```
        print(f"\n {key.upper().replace('_', ' ')}:")
```

```
        for k, v in value.items():
```

```
            if isinstance(v, dict):
```

```
                print(f"    {k}:")
```

```
                for kk, vv in v.items():
```

```
                    print(f"        {kk:20s}: {vv}")
```

```
            else:
```

```
                print(f"    {k:25s}: {v}")
```

```
        else:
```

```
            print(f" {key:30s}: {value}")
```

```
print("="*70)
```

Краткие выводы

```
print("\n    КЛЮЧЕВЫЕ ВЫВОДЫ:")
```

```
print("    1. Глубокие сети лучше аппроксимируют сложные зависимости, но требуют больше данных")
```

```
print("    2. Широкие сети быстрее обучаются, но могут переобучаться при малых выборках")
```

```
print("    3. Баланс глубины и ширины часто дает наилучший компромисс точность/сложность")
```

```
print(" 4. Выбор архитектуры должен учитывать не только точность, но и  
вычислительные ресурсы")  
print(" 5. Для химических задач важна интерпретируемость – иногда простая модель  
предпочтительнее")
```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность сравнения архитектур

1. Убедитесь, что все архитектуры обучаются без ошибок и завершаются за разумное время
2. Проверьте, что количество параметров модели соответствует конфигурации (сумма произведений размеров слоев)
3. Зафиксируйте, какая архитектура показала наилучший баланс точности и сложности

Тест 2: Влияние критерия выбора

1. Измените `selection_criterion` в Ячейке 9 на другой вариант
2. Перезапустите анализ и сравните, как изменится выбор лучшей архитектуры
3. Проанализируйте, в каких практических ситуациях предпочтителен каждый из критериев

Тест 3: Интерпретация прогноза

1. В Ячейке 10 задайте экстремальные значения параметров (на границах диапазона)
2. Оцените, насколько прогноз остается физически осмысленным
3. Сформулируйте вывод: как сложность архитектуры влияет на устойчивость прогноза при экстраполяции?

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, тип зависимости, набор тестируемых архитектур, критерий выбора)
4. **Разработанные документы:**
 - Скриншоты: графики компромиссов (точность/сложность, точность/время), кривые обучения для всех архитектур, столбчатая диаграмма финальных метрик
 - Сводная таблица результатов для всех архитектур (R^2 , RMSE, MAE, количество параметров, время обучения)
 - Блок анализа выбора оптимальной архитектуры с обоснованием критерия
 - Результат прогноза для новых условий с физико-химической интерпретацией
5. **Выводы** (4–5 пунктов):
 - Как глубина сети влияет на способность моделировать сложные нелинейные зависимости?
 - В каких случаях предпочтительнее широкая сеть, а в каких – глубокая?
 - Почему количество параметров модели важно учитывать при выборе архитектуры для реальных задач?
 - Как критерий выбора архитектуры зависит от практических ограничений (время обучения, доступные вычислительные ресурсы, требования к интерпретируемости)?
 - Практические рекомендации по выбору архитектуры нейронной сети для задач оптимизации химико-технологических процессов

6. **Приложение** (ссылка на Google Colab или исходный код)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему увеличение глубины нейронной сети не всегда приводит к улучшению качества модели, и какие проблемы могут возникнуть при обучении очень глубоких сетей?

Вопрос 2:

Как количество обучаемых параметров модели связано с риском переобучения, и почему это особенно критично для задач с ограниченным объемом экспериментальных данных?

Вопрос 3:

В чем заключается компромисс между точностью модели и ее вычислительной сложностью, и как этот компромисс влияет на выбор архитектуры в промышленных приложениях?

Вопрос 4:

Почему для задач химической технологии может быть предпочтительна более простая и интерпретируемая модель, даже если сложная архитектура демонстрирует немного лучшую точность?

Вопрос 5:

Какие дополнительные метрики или стратегии (например, кросс-валидация, тест на устойчивость к шуму, анализ чувствительности) можно использовать для более надежного сравнения архитектур в реальных проектах?

Лабораторная работа №9

«Итоговое моделирование: полный цикл нейросетевого анализа технологического процесса»

Цель: Закрепить комплексные навыки построения, обучения, валидации и интерпретации нейросетевой модели через выполнение сквозного проекта – от подготовки данных до формулировки технологических выводов, сформировать готовность к самостоятельному применению инструментов нейросетевого моделирования в профессиональной деятельности химика-технолога.

Что нужно сделать:

- Открыть Google Colab (<https://colab.research.google.com/>)
- Создать новый блокнот
- Скопировать код из методических указаний в ячейки Colab
- Запустить и протестировать программу
- Подготовить отчет о проделанной работе в файле .docx

1. ШАГ 1: СОЗДАНИЕ НОВОЙ ТЕТРАДИ В GOOGLE COLAB

2. Перейдите на сайт: <https://colab.research.google.com/>
3. Нажмите «Новый блокнот»
4. Назовите тетрадь: Лабораторная 9: Итоговый проект

5. ШАГ 2: КОПИРОВАНИЕ КОДА В ЯЧЕЙКИ COLAB

Ячейка 1: Заголовок программы

Создайте первую ячейку, нажмите на значок «Текст», и скопируйте этот текст:

Лабораторная работа №9

Итоговый проект: полный цикл нейросетевого моделирования

****Студент:**** [Ваша фамилия и имя]

****Дата:**** [Сегодняшняя дата]

****Вариант:**** [Номер варианта]

****Задание:****

Выполнить сквозной проект нейросетевого моделирования, который включает:

1. Постановку задачи и анализ предметной области (конкретный химико-технологический процесс)
2. Подготовку данных: генерация/загрузка, очистка, кодирование, стандартизация
3. Выбор и обоснование архитектуры нейронной сети
4. Настройку гиперпараметров обучения и применение методов регуляризации
5. Обучение модели, валидацию и оценку качества на независимых данных
6. Прогнозирование для новых условий и интерпретацию результатов в контексте технологии
7. Формулировку выводов и рекомендаций для практического применения

****Критерии успешного выполнения:****

- Корректность реализации всех этапов пайплайна
 - Обоснованность принятых решений (архитектура, гиперпараметры, методы предобработки)
 - Качество интерпретации результатов в терминах химической технологии
 - Профессиональное оформление отчета и аргументированность при защите
- Г8в

Ячейка 2: Импорт библиотек

Создайте новую ячейку «Код» и скопируйте:

```
# Импортируем нужные библиотеки
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam, SGD
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,
median_absolute_error
import joblib
import json
from datetime import datetime

print(" Все библиотеки загружены!")
print(f"Версия TensorFlow: {tf.__version__}")
print(" Keras: создание и обучение моделей")
print(" sklearn: предобработка, метрики, валидация")
print(" matplotlib/seaborn: визуализация")
print(" joblib/json: сохранение артефактов")
```

Ячейка 3: Постановка задачи и генерация данных (ЗАДАНИЕ 1)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 1: ПОСТАНОВКА ЗАДАЧИ И ГЕНЕРАЦИЯ ДАННЫХ
=====

# ===== ВАШ ВАРИАНТ: выбор технологического процесса =====
# Вариант 1: Синтез метанола (прогноз конверсии CO)
# Вариант 2: Каталитический крекинг (прогноз выхода бензина)
# Вариант 3: Полимеризация этилена (прогноз молекулярной массы)

process_name = 'methanol_synthesis' # Измените согласно варианту!

# Параметры генерации данных
np.random.seed(42)
n_samples = 500
noise_level = 2.5

print(f" Технологический процесс: {process_name}")
print("-"*60)

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: сгенерировать данные, соответствующие выбранному процессу
```

Подсказка:

- Определите входные параметры процесса (температура, давление, концентрации, время)

- Задайте физически обоснованную целевую функцию с нелинейностями

- Добавьте реалистичный шум и, опционально, пропущенные значения

Ваш код ниже:

```
if process_name == 'methanol_synthesis':
```

```
    # Синтез метанола:  $\text{CO} + 2\text{H}_2 \rightarrow \text{CH}_3\text{OH}$ 
```

```
    # Параметры: T, P, соотношение H2/CO, пространственная скорость
```

```
    T = np.random.uniform(200, 300, n_samples)    # Температура, °C
```

```
    P = np.random.uniform(50, 100, n_samples)    # Давление, бар
```

```
    ratio = np.random.uniform(1.8, 2.5, n_samples)    # Соотношение H2/CO
```

```
    GHSV = np.random.uniform(5000, 15000, n_samples)    # Пространственная скорость, ч-1
```

```
    # Целевая: конверсия CO (%) с нелинейной зависимостью
```

```
    conversion = (  
        40 + 0.15*(T-250) + 0.4*(P-75) + 8*(ratio-2.1) - 0.002*(T-250)**2 -  
        0.001*(P-75)*(T-250) - 0.00003*GHSV +  
        np.random.normal(0, noise_level, n_samples)  
    )
```

```
    feature_names = ['T_C', 'P_bar', 'H2_CO_ratio', 'GHSV_h']
```

```
elif process_name == 'catalytic_cracking':
```

```
    # Каталитический крекинг: выход бензина (%)
```

```
    T = np.random.uniform(480, 540, n_samples)    # Температура, °C
```

```
    catalyst_age = np.random.uniform(0, 100, n_samples)    # Возраст катализатора, %
```

```
    feed_rate = np.random.uniform(10, 30, n_samples)    # Подача сырья, т/ч
```

```
    gasoline_yield = (  
        55 + 0.2*(T-510) - 0.3*catalyst_age - 0.4*(feed_rate-20) -  
        0.001*(T-510)**2 + 0.002*catalyst_age*(T-510) +  
        np.random.normal(0, noise_level, n_samples)  
    )
```

```
    gasoline_yield = np.clip(gasoline_yield, 20, 75)
```

```
    feature_names = ['T_C', 'catalyst_age_pct', 'feed_rate_tph']
```

```
else: # polymerization
```

```
    # Полимеризация этилена: молекулярная масса (г/моль)
```

```
    T = np.random.uniform(60, 100, n_samples)    # Температура, °C
```

```
    P = np.random.uniform(10, 40, n_samples)    # Давление, бар
```

```
    catalyst_conc = np.random.uniform(0.01, 0.1, n_samples)    # Концентрация катализатора
```

```
    Mw = (  
        25000 + 200*(T-80) - 500*(P-25) + 50000*catalyst_conc -  
        5*(T-80)**2 + 200*(P-25)*catalyst_conc +  
        np.random.normal(0, noise_level*100, n_samples)  
    )
```

```
    Mw = np.clip(Mw, 10000, 100000)
```

```
    feature_names = ['T_C', 'P_bar', 'catalyst_conc']
```

```

# Формирование DataFrame
df = pd.DataFrame({name: data for name, data in zip(feature_names, [T, P, ratio,
GHSV][:len(feature_names)])})
target_name = 'CO_conversion_pct' if process_name == 'methanol_synthesis' else
('gasoline_yield_pct' if process_name == 'catalytic_cracking' else 'Mw_gmol')
df[target_name] = conversion if process_name == 'methanol_synthesis' else (gasoline_yield if
process_name == 'catalytic_cracking' else Mw)

# Добавление пропусков (5% для имитации реальных данных)
missing_cols = np.random.choice(feature_names, size=2, replace=False)
for col in missing_cols:
    missing_idx = np.random.choice(n_samples, size=int(n_samples*0.05), replace=False)
    df.loc[missing_idx, col] = np.nan

print(f"    Сгенерировано {n_samples} образцов")
print(f"    Признаки: {feature_names}")
print(f"    Целевая переменная: {target_name}")
print(f"    Диапазон цели: {df[target_name].min():.1f} – {df[target_name].max():.1f}")
print(f"    Пропущенные значения: {df.isnull().sum().sum()}")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 4: Предобработка данных (ЗАДАНИЕ 2)

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ЗАДАНИЕ 2: ПОЛНЫЙ ПАЙПЛАЙН ПРЕДОБРАБОТКИ
=====

```

```

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: реализовать корректный пайплайн предобработки данных

# Подсказка:
# 1. Разделить данные ДО применения fit() преобразователей
# 2. Обработать пропуски (SimpleImputer)
# 3. Стандартизировать числовые признаки (StandardScaler)
# 4. Сохранить все преобразователи для воспроизводимости

# Ваш код ниже:

# 1. Разделение на признаки и целевую переменную
X = df[feature_names].copy()
y = df[target_name].copy()

# 2. Разделение на train/test (стратификация не нужна для регрессии)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print(f"    Обучающая выборка: {X_train.shape[0]}, Тестовая: {X_test.shape[0]}")

# 3. Обработка пропусков

```

```

imputer = SimpleImputer(strategy='mean')
X_train_imp = pd.DataFrame(imputer.fit_transform(X_train), columns=feature_names)
X_test_imp = pd.DataFrame(imputer.transform(X_test), columns=feature_names)
print(f"    Пропуски обработаны (осталось: {X_train_imp.isnull().sum().sum()})")

# 4. Стандартизация числовых признаков
scaler = StandardScaler()
X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train_imp), columns=feature_names)
X_test_scaled = pd.DataFrame(scaler.transform(X_test_imp), columns=feature_names)
print(f"    Стандартизация выполнена (mean={X_train_scaled.mean().mean():.3f},
std={X_train_scaled.std().mean():.3f})")

# 5. Сохранение преобразователей
joblib.dump({'imputer': imputer, 'scaler': scaler}, 'preprocessors.pkl')
print("    Преобразователи сохранены: preprocessors.pkl")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 5: Выбор и обоснование архитектуры модели (ЗАДАНИЕ 3)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 3: ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ МОДЕЛИ =====

```

# ===== ВАШ ВАРИАНТ: базовая конфигурация архитектуры =====
# Вариант 1: Консервативная (2 слоя, умеренное количество нейронов)
# Вариант 2: Сбалансированная (3 слоя, адаптивная ширина)
# Вариант 3: Исследовательская (4+ слоя, с Dropout и L2)

architecture_style = 'balanced' # Измените согласно варианту!

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: создать функцию построения модели с обоснованием выбора архитектуры

# Подсказка:
# - Обоснуйте выбор количества слоев и нейронов исходя из сложности задачи
# - Добавьте регуляризацию (Dropout, L2) при необходимости
# - Используйте функциональный API для сложных архитектур (опционально)

# Ваш код ниже:

def create_final_model(input_dim, style='balanced'):
    """
    Создает финальную модель с архитектурой, соответствующей стилю.
    """
    model = Sequential()

    if style == 'conservative':
        # Простая архитектура для интерпретируемости
        model.add(Dense(16, activation='relu', input_shape=(input_dim,),
kernel_regularizer=l2(0.001)))
        model.add(Dense(8, activation='relu', kernel_regularizer=l2(0.001)))

```

```

elif style == 'balanced':
    # Сбалансированная архитектура
    model.add(Dense(32, activation='relu', input_shape=(input_dim,),
kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.2))
    model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.001)))
    model.add(Dropout(0.2))

else: # research
    # Исследовательская архитектура с регуляризацией
    model.add(Dense(48, activation='relu', input_shape=(input_dim,),
kernel_regularizer=l2(0.0005)))
    model.add(Dropout(0.3))
    model.add(Dense(24, activation='relu', kernel_regularizer=l2(0.0005)))
    model.add(Dropout(0.3))
    model.add(Dense(12, activation='relu', kernel_regularizer=l2(0.0005)))

# Выходной слой (линейный для регрессии)
model.add(Dense(1, activation='linear'))

return model

# Создание модели
input_dim = X_train_scaled.shape[1]
model_final = create_final_model(input_dim, style=architecture_style)

print(f"    Создана модель в стиле '{architecture_style}'")
print(f"    Входная размерность: {input_dim}")
print(f"    Количество параметров: {model_final.count_params():,}")
model_final.summary()

```

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 6: Настройка обучения и запуск тренировки (ЗАДАНИЕ 4)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 4: НАСТРОЙКА И ЗАПУСК ОБУЧЕНИЯ =====

===== ВАШ ВАРИАНТ: гиперпараметры обучения =====

Вариант 1: Консервативные (малый LR, много эпох)

Вариант 2: Стандартные (Adam, умеренные настройки)

Вариант 3: Агрессивные (большой LR, EarlyStopping)

training_config = 'standard' # Измените согласно варианту!

===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

ЗАДАНИЕ: настроить оптимизатор, callbacks и запустить обучение

Подсказка:

- Выберите оптимизатор и learning_rate в зависимости от config

- Настройте EarlyStopping и/или ReduceLROnPlateau

- Зафиксируйте время обучения для анализа эффективности

Ваш код ниже:

Компиляция модели

if training_config == 'conservative':

optimizer = SGD(learning_rate=0.001, momentum=0.9)

epochs = 300

patience = 30

elif training_config == 'standard':

optimizer = Adam(learning_rate=0.001)

epochs = 200

patience = 20

else: # aggressive

optimizer = Adam(learning_rate=0.01)

epochs = 150

patience = 10

model_final.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

Callbacks

callbacks = [

EarlyStopping(monitor='val_loss', patience=patience, restore_best_weights=True, verbose=0),

ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=1e-7, verbose=0)

]

Обучение с замером времени

import time

start_time = time.time()

history = model_final.fit(

X_train_scaled, y_train,

epochs=epochs,

batch_size=16,

validation_split=0.2,

callbacks=callbacks,

verbose=1 # Показывать прогресс для итогового проекта

)

train_time = time.time() - start_time

print(f"\n Обучение завершено за {train_time:.1f} секунд")

print(f" Обучено эпох: {len(history.history['loss'])} из {epochs}")

print(f" Финальные потери: train={history.history['loss'][-1]:.3f},

val={history.history['val_loss'][-1]:.3f}")

===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

Ячейка 7: Комплексная оценка качества модели (ЗАДАНИЕ 5)

Создайте новую ячейку с кодом и скопируйте:

===== ЗАДАНИЕ 5: КОМПЛЕКСНАЯ ОЦЕНКА КАЧЕСТВА =====

```

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
# ЗАДАНИЕ: рассчитать и визуализировать полный набор метрик качества

# Подсказка:
# - Используйте несколько метрик:  $R^2$ , RMSE, MAE, MedAE
# - Постройте графики: факт/прогноз, остатки, кривые обучения
# - Проанализируйте распределение ошибок

# Ваш код ниже:

# Прогнозы на тестовой выборке
y_test_pred = model_final.predict(X_test_scaled, verbose=0).flatten()

# Расчет метрик
metrics = {
    'R2': r2_score(y_test, y_test_pred),
    'RMSE': np.sqrt(mean_squared_error(y_test, y_test_pred)),
    'MAE': mean_absolute_error(y_test, y_test_pred),
    'MedAE': median_absolute_error(y_test, y_test_pred),
    'MAPE': np.mean(np.abs((y_test - y_test_pred) / y_test)) * 100 # в %
}

# Вывод метрик
print("\n" + "="*70)
print("    КОМПЛЕКСНАЯ ОЦЕНКА КАЧЕСТВА МОДЕЛИ")
print("="*70)
print(f"{'Метрика':<25} {'Значение':>20}")
print("-"*70)
print(f"{'R² (коэфф. детерминации)':<25} {metrics['R2']:>20.4f}")
print(f"{'RMSE (среднеквадратичная)':<25} {metrics['RMSE']:>20.2f}")
print(f"{'MAE (средняя абсолютная)':<25} {metrics['MAE']:>20.2f}")
print(f"{'MedAE (медианная абсолютная)':<25} {metrics['MedAE']:>20.2f}")
print(f"{'MAPE (средняя % ошибка)':<25} {metrics['MAPE']:>20.2f}%")
print("="*70)

# Визуализация
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# График 1: Факт против Прогноза
axes[0, 0].scatter(y_test, y_test_pred, alpha=0.7, edgecolors='black', s=30)
axes[0, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2)
axes[0, 0].set_xlabel('Фактическое значение')
axes[0, 0].set_ylabel('Прогноз модели')
axes[0, 0].set_title(f'Факт vs Прогноз ( $R^2 = {metrics["R2"]:.3f}$ )')
axes[0, 0].grid(True, alpha=0.3)

# График 2: Распределение остатков
residuals = y_test - y_test_pred
axes[0, 1].hist(residuals, bins=30, edgecolors='black', alpha=0.7)
axes[0, 1].axvline(0, color='red', linestyle='--', linewidth=2)
axes[0, 1].set_xlabel('Остаток (факт - прогноз)')

```

```

axes[0, 1].set_ylabel('Частота')
axes[0, 1].set_title(f'Распределение ошибок (mean={residuals.mean():.2f},
std={residuals.std():.2f})')
axes[0, 1].grid(True, alpha=0.3)

# График 3: Остатки от прогноза (поиск систематических ошибок)
axes[1, 0].scatter(y_test_pred, residuals, alpha=0.7, edgecolors='black', s=30)
axes[1, 0].axhline(0, color='red', linestyle='--', linewidth=1)
axes[1, 0].set_xlabel('Прогнозируемое значение')
axes[1, 0].set_ylabel('Остаток')
axes[1, 0].set_title('Анализ систематических ошибок')
axes[1, 0].grid(True, alpha=0.3)

# График 4: Кривые обучения
axes[1, 1].plot(history.history['loss'], label='Train Loss', linewidth=2)
axes[1, 1].plot(history.history['val_loss'], label='Val Loss', linewidth=2)
axes[1, 1].set_xlabel('Эпоха')
axes[1, 1].set_ylabel('MSE (потеря)')
axes[1, 1].set_title('Динамика обучения')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].set_yscale('log')

plt.tight_layout()
plt.show()

# Интерпретация качества
print("\n ИНТЕРПРЕТАЦИЯ КАЧЕСТВА:")
if metrics['R2'] > 0.9:
    print(" Отличное качество: модель объясняет >90% дисперсии данных")
elif metrics['R2'] > 0.75:
    print(" Хорошее качество: модель пригодна для прогнозирования и оптимизации")
elif metrics['R2'] > 0.6:
    print(" Умеренное качество: модель может использоваться для скрининга, но
    требует доработки")
else:
    print(" Низкое качество: необходима переработка архитектуры или данных")

if metrics['MAPE'] < 5:
    print(" Средняя ошибка прогноза <5% – высокая точность для технологических
    расчетов")
elif metrics['MAPE'] < 10:
    print(" Средняя ошибка 5-10% – приемлемо для предварительной оптимизации")
else:
    print(" Средняя ошибка >10% – требуется улучшение модели или сбор
    дополнительных данных")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 8: Прогнозирование и анализ чувствительности (ЗАДАНИЕ 6)

Создайте новую ячейку с кодом и скопируйте:

```
# ===== ЗАДАНИЕ 6: ПРОГНОЗ И АНАЛИЗ ЧУВСТВИТЕЛЬНОСТИ  
=====
```

```
# ===== ВАШ ВАРИАНТ: условия для прогноза и анализа =====
```

```
# Вариант 1: Базовые условия + анализ по одному фактору
```

```
# Вариант 2: Экстремальные условия + многофакторный анализ
```

```
# Вариант 3: Оптимизационный сценарий + поиск оптимума
```

```
analysis_type = 'sensitivity' # Измените: 'baseline', 'extreme', 'optimization'
```

```
print(f"\n РЕЖИМ АНАЛИЗА: {analysis_type}")
```

```
print("="*70)
```

```
# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====
```

```
# ЗАДАНИЕ: реализовать прогноз и анализ чувствительности модели
```

```
# Подсказка:
```

```
# - Подготовьте новые данные с применением тех же преобразователей
```

```
# - Для анализа чувствительности: варьировать один параметр при фиксированных  
остальных
```

```
# - Визуализировать зависимость прогноза от варьируемого параметра
```

```
# Ваш код ниже:
```

```
# Базовые условия для прогноза (измените согласно варианту!)
```

```
baseline_params = {
```

```
    'T_C': 250 if process_name == 'methanol_synthesis' else (510 if process_name ==  
'catalytic_cracking' else 80),
```

```
    'P_bar': 75 if process_name in ['methanol_synthesis', 'polymerization'] else None,
```

```
    'H2_CO_ratio': 2.1 if process_name == 'methanol_synthesis' else None,
```

```
    'GHSV_h': 10000 if process_name == 'methanol_synthesis' else None,
```

```
    'catalyst_age_pct': 50 if process_name == 'catalytic_cracking' else None,
```

```
    'feed_rate_tph': 20 if process_name == 'catalytic_cracking' else None,
```

```
    'catalyst_conc': 0.05 if process_name == 'polymerization' else None
```

```
}
```

```
# Фильтрация None и создание вектора
```

```
X_baseline = np.array([[baseline_params[name] for name in feature_names]])
```

```
X_baseline_scaled = scaler.transform(X_baseline)
```

```
# Прогноз для базовых условий
```

```
pred_baseline = model_final.predict(X_baseline_scaled, verbose=0)[0, 0]
```

```
print(f" Базовые условия: {dict(zip(feature_names, X_baseline[0]))}")
```

```
print(f" → Прогноз {target_name}: {pred_baseline:.2f}")
```

```
# Анализ чувствительности (варьирование одного параметра)
```

```
if analysis_type == 'sensitivity' and len(feature_names) >= 2:
```

```
    var_param = feature_names[0] # Варьируемый параметр
```

```
    var_range = np.linspace(X_train_scaled[var_param].quantile(0.1),
```

```
                            X_train_scaled[var_param].quantile(0.9), 20)
```

```

predictions_sens = []
for val in var_range:
    X_test_sens = X_baseline.copy()
    X_test_sens[0, feature_names.index(var_param)] = val
    X_test_sens_scaled = scaler.transform(X_test_sens)
    pred = model_final.predict(X_test_sens_scaled, verbose=0)[0, 0]
    predictions_sens.append(pred)

# Визуализация
plt.figure(figsize=(8, 5))
plt.plot(var_range, predictions_sens, linewidth=2, color='green', marker='o')
plt.xlabel(f'{var_param} (стандартизированное значение)')
plt.ylabel(f'Прогноз {target_name}')
plt.title(f'Чувствительность модели к {var_param}')
plt.grid(True, alpha=0.3)
plt.show()

# Интерпретация
sens_slope = (predictions_sens[-1] - predictions_sens[0]) / (var_range[-1] - var_range[0])
print(f'    Чувствительность к {var_param}: {sens_slope:.3f} единиц цели / единицу параметра")
if abs(sens_slope) > 1:
    print("    Высокая чувствительность – параметр критичен для управления процессом")
else:
    print("    Умеренная чувствительность – параметр может использоваться для тонкой настройки")

# Проверка физической осмысленности
target_min, target_max = df[target_name].min(), df[target_name].max()
if target_min <= pred_baseline <= target_max:
    print("    Прогноз в физически допустимом диапазоне")
else:
    print("    Прогноз выходит за границы обучающих данных – возможна экстраполяция")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

Ячейка 9: Сохранение артефактов и формирование отчета (ЗАДАНИЕ 7)

Создайте новую ячейку с кодом и скопируйте:

```

# ===== ЗАДАНИЕ 7: СОХРАНЕНИЕ АРТЕФАКТОВ И ОТЧЕТА
=====

```

```

# ===== ВАША РАБОТА НАЧИНАЕТСЯ ЗДЕСЬ =====

```

ЗАДАНИЕ: сохранить все компоненты проекта для воспроизводимости

Подсказка:

- Сохраните модель, преобразователи, метрики и конфигурацию

- Сформируйте структурированный JSON-отчет

- Подготовьте инструкцию по загрузке и использованию артефактов

Ваш код ниже:

1. Сохранение модели

```
model_final.save('final_model.keras')
```

```
print("    Модель сохранена: final_model.keras")
```

2. Сохранение преобразователей (если не сохранено ранее)

```
if not os.path.exists('preprocessors.pkl'):
```

```
    joblib.dump({'imputer': imputer, 'scaler': scaler}, 'preprocessors.pkl')
```

```
    print("    Преобразователи сохранены: preprocessors.pkl")
```

3. Формирование итогового отчета

```
final_report = {
```

```
    'project_metadata': {
```

```
        'date': datetime.now().strftime("%d.%m.%Y %H:%M"),
```

```
        'student': '[Ваша фамилия]',
```

```
        'variant': '[Номер варианта]',
```

```
        'process_name': process_name,
```

```
        'target_variable': target_name
```

```
    },
```

```
    'data_summary': {
```

```
        'n_samples': n_samples,
```

```
        'n_features': len(feature_names),
```

```
        'feature_names': feature_names,
```

```
        'missing_values_handled': True,
```

```
        'train_test_split': '80/20'
```

```
    },
```

```
    'model_configuration': {
```

```
        'architecture_style': architecture_style,
```

```
        'training_config': training_config,
```

```
        'n_parameters': model_final.count_params(),
```

```
        'epochs_trained': len(history.history['loss']),
```

```
        'optimizer': optimizer.__class__.__name__,
```

```
        'regularization': 'L2 + Dropout' if architecture_style != 'conservative' else 'L2 only'
```

```
    },
```

```
    'performance_metrics': {k: float(v) for k, v in metrics.items()},
```

```
    'prediction_example': {
```

```
        'input_conditions': {name: float(val) for name, val in zip(feature_names, X_baseline[0])},
```

```
        'predicted_value': float(pred_baseline),
```

```
        'physical_range': [float(target_min), float(target_max)]
```

```
    },
```

```
    'artifacts': {
```

```
        'model_file': 'final_model.keras',
```

```
        'preprocessors_file': 'preprocessors.pkl',
```

```
        'training_history': 'history.json'
```

```
    }
```

```
}
```

Сохранение отчета

```
with open('final_report.json', 'w', encoding='utf-8') as f:
```

```
    json.dump(final_report, f, ensure_ascii=False, indent=2)
```

```

print("    Отчет сохранен: final_report.json")

# Сохранение истории обучения
with open('history.json', 'w') as f:
    json.dump(history.history, f)

# Вывод сводки
print("\n" + "="*70)

print("    ИТОГОВАЯ СВОДКА ПРОЕКТА")
print("="*70)
for section, content in final_report.items():
    print(f"\n{section.upper().replace('_', ' ')}:")
    if isinstance(content, dict):
        for key, value in content.items():
            print(f"    {key:30s}: {value}")
    else:
        print(f"    {content}")
print("="*70)

# Инструкция по воспроизведению
print("\n    ИНСТРУКЦИЯ ПО ВОСПРОИЗВЕДЕНИЮ РЕЗУЛЬТАТОВ:")
print("    1. Загрузите артефакты: final_model.keras, preprocessors.pkl")
print("    2. Подготовьте новые данные в том же формате, что и обучающие")
print("    3. Примените преобразователи: imputer.transform() → scaler.transform()")
print("    4. Загрузите модель: load_model('final_model.keras')")
print("    5. Получите прогноз: model.predict(preprocessed_data)")

print("\n    Для развертывания в промышленной системе рекомендуется:")
print("    - Протестировать модель на независимом наборе данных")
print("    - Реализовать мониторинг дрейфа данных (data drift)")
print("    - Настроить периодическое дообучение модели на новых данных")

# ===== ВАША РАБОТА ЗАКОНЧИЛАСЬ =====

```

6. ШАГ 3: ПРОВЕРКА РАБОТЫ ПРОГРАММЫ

Проведите следующие тесты:

Тест 1: Корректность полного пайплайна

1. Убедитесь, что все этапы выполняются последовательно без ошибок
2. Проверьте, что метрики качества на тестовой выборке адекватны ($R^2 > 0.6$ для учебного проекта)
3. Зафиксируйте время обучения и количество параметров модели

Тест 2: Воспроизводимость прогноза

1. Сохраните артефакты (модель, preprocessors)
2. В новом блоке кода загрузите их и повторите прогноз для тех же условий
3. Убедитесь, что результат совпадает с исходным (с точностью до численной погрешности)

Тест 3: Анализ чувствительности и интерпретация

1. Измените варьируемый параметр в анализе чувствительности
2. Оцените, соответствует ли направление изменения прогноза физико-химическим ожиданиям
3. Сформулируйте рекомендацию: как использовать эту чувствительность для управления процессом?

7. ШАГ 4: ПОДГОТОВКА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Структура отчета (формат Word):

1. **Титульный лист** (университет, кафедра, ФИО, группа, номер работы)
2. **Тема и цель работы** (2–3 предложения)
3. **Условия задания** (номер варианта, описание технологического процесса, ключевые параметры)
4. **Разработанные документы:**
 - Схема полного пайплайна моделирования (блок-схема или описание этапов)
 - Скриншоты: визуализация данных, кривые обучения, графики «Факт vs Прогноз», анализ остатков, чувствительность
 - Таблица комплексных метрик качества (R^2 , RMSE, MAE, MedAE, MAPE)
 - Результат прогноза для базовых условий с физико-химической интерпретацией
 - Сводная таблица артефактов проекта (файлы, версии, зависимости)
5. **Выводы** (5–6 пунктов):
 - Насколько успешно решена поставленная задача моделирования?
 - Какие этапы пайплайна оказались наиболее критичными для итогового качества?
 - Как выбор архитектуры и гиперпараметров повлиял на результат?
 - В чем заключаются ограничения разработанной модели?
 - Как результаты моделирования могут быть использованы для оптимизации реального технологического процесса?
 - Какие шаги необходимы для перевода модели из учебного прототипа в промышленную эксплуатацию?
6. **Приложение** (ссылка на Google Colab, исходный код, архив с артефактами)

8. КОНТРОЛЬНЫЕ ВОПРОСЫ ДЛЯ ОТЧЕТА И ЗАЩИТЫ

Вопрос 1:

Почему в итоговом проекте важно сохранять не только модель, но и все преобразователи данных (imputer, scaler)? К каким ошибкам может привести нарушение этого правила при прогнозировании на новых данных?

Вопрос 2:

Как интерпретировать значение $MAPE = 7.3\%$ в контексте управления химико-технологическим процессом? В каких случаях такая точность достаточна для принятия технологических решений, а в каких – нет?

Вопрос 3:

Почему анализ чувствительности модели к входным параметрам важен для химика-технолога, даже если модель демонстрирует высокое качество прогноза?

Вопрос 4:

Какие дополнительные проверки и валидации необходимо провести перед внедрением нейросетевой модели в систему поддержки принятия решений на реальном производстве?

Вопрос 5:

Как обеспечить долгосрочную актуальность модели в условиях изменения характеристик сырья, износа оборудования или модификации технологического регламента?

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
для обучающихся по организации и проведению самостоятельной работы
по дисциплине
**«НЕЙРОСЕТЕВОЕ МОДЕЛИРОВАНИЕ ТЕХНОЛОГИЧЕСКИХ
ПРОЦЕССОВ»**

Ставрополь, 2026

СОДЕРЖАНИЕ

1. Общие положения	3
2. Цель и задачи самостоятельной работы	4
3. Технологическая карта самостоятельной работы студента	5
4. Порядок выполнения самостоятельной работы студентом	5
4.1. Методические рекомендации по работе с учебной литературой	5
4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям	7
4.3. Методические рекомендации по самопроверке знаний	7
4.4. Методические рекомендации по написанию научных текстов (докладов, докладов, эссе, научных статей и т.д.)	7
4.5. Методические рекомендации по выполнению исследовательских проектов	10
4.6. Методические рекомендации по подготовке к экзаменам и зачетам	13
5. Контроль самостоятельной работы студентов	14
6. Список литературы для выполнения СРС	14

1. Общие положения

Самостоятельная работа - планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

- формирование и усвоение содержания лекционного материала на базе рекомендованной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки, документация TensorFlow/Keras);
- подготовка к лабораторным работам: изучение теоретического материала, предварительное выполнение заданий в среде Google Colab, оформление отчетов;
- написание докладов и рефератов по темам применения искусственного интеллекта в химической технологии;
- выполнение учебно-исследовательских работ: разработка нейросетевых моделей для конкретных химических процессов, анализ экспериментальных данных;
- подготовка практических разработок и рекомендаций по оптимизации режимов химико-технологических процессов на основе нейросетевого моделирования;
- компьютерный текущий самоконтроль и контроль успеваемости на базе электронных тестов и автоматизированных систем проверки кода;
- выполнение индивидуального творческого задания: сквозной проект нейросетевого моделирования от сбора данных до интерпретации результатов;
- подготовка к промежуточной аттестации (зачету).

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- **Подготовительный:** определение целей, составление программы, подготовка методического обеспечения (шаблоны ноутбуков, наборы данных), подготовка оборудования (доступ к Google Colab, установка необходимых библиотек);
- **Основной:** реализация программы, использование приемов поиска информации (научные статьи, документация библиотек), усвоение, переработка, применение, передача знаний, фиксирование результатов (сохранение моделей, отчетов), самоорганизация процесса работы;
- **Заключительный:** оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда.

Самостоятельная работа по дисциплине «Нейросетевое моделирование технологических процессов» направлена на формирование следующих компетенций:

Код	Формулировка:
УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач
ОПК-3	Способен применять и модифицировать математические модели для решения задач в области профессиональной деятельности
ОПК-5	Способен проводить физико-химический анализ веществ и материалов, интерпретировать результаты экспериментальных исследований
ОПК-6	Способен разрабатывать и оптимизировать технологические процессы и схемы производств, оценивать их эффективность и безопасность

2. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего бакалавра по направлению подготовки 04.03.01 «Химия».

Целью самостоятельной работы студентов является овладение фундаментальными знаниями в области применения искусственных нейронных сетей для анализа, математического описания, оптимизации и предиктивного управления химико-технологическими процессами, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3. Технологическая карта самостоятельной работы студента

3. Технологическая карта самостоятельной работы студента					
Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
3 семестр					
ИД-1, ИД-2, ИД-3 УК-1; ИД-1, ИД-2, ИД-3 ОПК-3; ИД-1, ИД-2, ИД-3 ОПК-5; ИД-1, ИД-2, ИД-3 ОПК-6	Самостоятельное изучение литературы и источников по темам: математические основы ИНС, архитектуры сетей, алгоритмы обучения	Собеседование, тестирование	18	18	36
ИД-1, ИД-2, ИД-3 УК-1; ИД-1, ИД-2, ИД-3 ОПК-3; ИД-1, ИД-2, ИД-3 ОПК-5; ИД-1, ИД-2, ИД-3 ОПК-6	Подготовка к лабораторным занятиям: изучение теории, предварительное выполнение кода в Google Colab, оформление отчетов	Защита лабораторной работы, оценка отчета	24	24	48
ИД-1, ИД-2, ИД-3 УК-1; ИД-1, ИД-2, ИД-3 ОПК-3; ИД-1, ИД-2, ИД-3 ОПК-5; ИД-1, ИД-2, ИД-3 ОПК-6	Написание доклада/реферата по применению нейросетей в химической технологии (спектральный анализ, оптимизация процессов, контроль качества)	Защита доклада, оценка письменной работы	6	6	12
ИД-1, ИД-2, ИД-3 УК-1; ИД-1, ИД-2, ИД-3 ОПК-3; ИД-1, ИД-2, ИД-3 ОПК-5; ИД-1, ИД-2, ИД-3 ОПК-6	Выполнение индивидуального творческого задания: сквозной проект нейросетевого моделирования химико-технологического процесса	Защита проекта, оценка отчета и презентации	6	6	12
Итого за 3 семестр			54	54	108
Итого			54	54	108

4. Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с литературой по дисциплине «Нейросетевое моделирование технологических процессов» необходимо подбирать источники, учиться правильно их читать и вести записи. Для подбора литературы используются:

– электронно-библиотечные системы СКФУ («Университетская библиотека онлайн», IPRbooks); – научная электронная библиотека eLIBRARY.RU; – документация официальных ресурсов: TensorFlow.org, Keras.io, scikit-learn.org; – открытые репозитории кода: GitHub, Kaggle (для изучения примеров реализации).

Важно помнить, что рациональные навыки работы с литературой – это экономия времени и повышение качества усвоения материала.

Рекомендации по изучению материала:

1. **Последовательность изучения:** переходите к следующему вопросу только после уяснения предыдущего. Все математические выкладки (градиентный спуск, функции потерь, правила обратного распространения ошибки) рекомендуется воспроизводить на бумаге самостоятельно, даже если они опущены в учебнике.

2. **Работа с определениями:** особое внимание уделяйте основным понятиям курса (нейрон, функция активации, эпоха, батч, переобучение, регуляризация). Студент должен не только запомнить определения, но и уметь привести примеры их проявления в контексте химических задач.

3. **Конспектирование:** полезно составлять опорные конспекты лекций, дополняя их материалами из учебников. На полях отмечайте вопросы для консультации с преподавателем. Выводы и ключевые формулы выделяйте для лучшего запоминания.

4. **Лист опорных сигналов:** составьте справочный лист с важнейшими формулами, архитектурами сетей, гиперпараметрами и их типичными значениями. Такой лист служит постоянным справочником при выполнении лабораторных работ.

5. **Установки при чтении научного текста:**

- *Информационно-поисковая:* найти конкретную информацию (например, описание функции активации ReLU);
- *Усваивающая:* полностью осознать и запомнить материал (например, алгоритм обратного распространения ошибки);
- *Аналитико-критическая:* критически осмыслить материал, сравнить подходы разных авторов;
- *Творческая:* использовать прочитанное как основу для собственных исследований (например, адаптация архитектуры сети под конкретную химическую задачу).

Виды систематизированной записи прочитанного:

- **Аннотирование** – краткое описание статьи или главы книги, ее содержания и назначения;
- **Планирование** – логическая структура материала, раскрывающая последовательность изложения;
- **Тезирование** – лаконичное воспроизведение основных утверждений автора;
- **Цитирование** – дословное выписывание ключевых фрагментов с указанием источника;
- **Конспектирование** – краткое последовательное изложение содержания с сохранением логики автора.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные термины (например, «ко-адаптация нейронов», «затухающий градиент»).

2. Выделите главное, составьте план раздела.
 3. Сформулируйте основные положения своими словами, отметьте аргументацию автора.
 4. Конспектируйте материал, следуя пунктам плана. Старайтесь выражать мысли своими словами, записи ведите четко и ясно.
 5. Грамотно оформляйте цитаты: указывайте источник, страницу, автора.
 6. В тексте конспекта приводите не только тезисы, но и их доказательства, примеры из химической технологии.
 7. Стремитесь к емкости каждого предложения. Мысли излагайте кратко, заботясь о стиле и выразительности.
 8. Оставляйте поля для уточнений и дополнений при последующем изучении.
- Овладение навыками конспектирования требует целеустремленности и повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Лабораторные работы по дисциплине выполняются в облачной среде Google Colab с использованием языка программирования Python и библиотек машинного обучения (TensorFlow/Keras, scikit-learn, pandas, matplotlib).

Для того чтобы лабораторные занятия приносили максимальную пользу, необходимо:

1. **Предварительное изучение теории:** перед выполнением лабораторной работы внимательно изучите соответствующий раздел лекционного курса и рекомендованной литературы. Уясните цель работы, теоретические основы используемого метода, последовательность действий.
2. **Знакомство с шаблоном кода:** большинство лабораторных работ предоставлены в виде готовых ноутбуков (.ipynb) с комментариями. Перед занятием просмотрите код, уясните назначение каждой ячейки, подготовьте вопросы по непонятным фрагментам.
3. **Предварительный запуск кода:** по возможности выполните предварительный запуск ноутбука в Google Colab до занятия. Это позволит выявить технические проблемы (версии библиотек, доступ к ресурсам) и сосредоточиться на содержательной части во время занятия.
4. **Подготовка данных для варианта:** если лабораторная работа предусматривает индивидуальные варианты (диапазоны параметров, тип нелинейности, архитектура сети), заранее подготовьте свои значения и внесите их в соответствующие ячейки кода.
5. **Планирование отчета:** заранее ознакомьтесь с требованиями к оформлению отчета. Подготовьте шаблоны для скриншотов, таблиц с метриками, графиков. Это сэкономит время при оформлении результатов.

При самостоятельном выполнении заданий:

- обосновывайте каждый этап решения, исходя из теоретических положений курса (например, почему выбрана функция активации ReLU, почему применена стандартизация данных);
- если видите несколько путей решения задачи (например, разные архитектуры сети), сравните их и выберите наиболее рациональный, аргументируя выбор;
- до начала вычислений составьте краткий план решения: какие данные подготовить, какую модель создать, какие метрики рассчитать;
- решение излагайте подробно, вычисления располагайте в строгом порядке, отделяя вспомогательные вычисления от основных;

- решения сопровождайте комментариями, схемами архитектур, графиками зависимостей;
- доводите решение до окончательного логического вывода, которого требует условие задачи (например, «модель пригодна для прогнозирования», «требуется доработка архитектуры»);
- проверяйте полученный результат: физическая осмысленность прогноза, соответствие диапазонам параметров, анализ остатков модели.

4.3. Методические рекомендации по самопроверке знаний

После изучения темы по записям в конспекте и учебнику, а также после выполнения соответствующих лабораторных работ, студенту рекомендуется провести самопроверку усвоенных знаний.

Рекомендуемые приемы самопроверки:

1. **Ответы на контрольные вопросы:** по каждой теме курса предусмотрены контрольные вопросы (в методических указаниях к лабораторным работам, в фонде оценочных средств). Ответьте на них письменно или устно, затем сверьтесь с материалами курса.
2. **Решение типовых задач:** попробуйте самостоятельно решить задачу на построение простой нейронной сети «с нуля» (без использования готового шаблона): сгенерировать данные, создать модель, обучить, оценить качество.
3. **Тестирование:** пройдите электронные тесты по темам курса (при наличии в электронной информационно-образовательной среде СКФУ). Проанализируйте ошибки, повторите соответствующие разделы.
4. **Взаимопроверка:** обсудите решения с одноклассниками, сравните подходы к выполнению лабораторных работ, аргументируйте свой выбор архитектуры или гиперпараметров.
5. **Практическая проверка:** критерием усвоения материала является способность выполнить лабораторную работу без подсказок, объяснить каждый шаг кода, интерпретировать результаты.

Если в ходе самопроверки выявлены пробелы:

– вернитесь к соответствующему разделу лекций или учебника; – выполните дополнительные упражнения (например, измените гиперпараметры и проанализируйте влияние на качество модели); – обратитесь к преподавателю на консультации с конкретными вопросами.

4.4. Методические рекомендации по написанию научных текстов (докладов, докладов, эссе, научных статей и т.д.)

Перед тем как приступить к написанию научного текста, важно определить его истинную цель – это поможет разумно распределить силы и время.

Этапы подготовки доклада/реферата:

1. **Выбор темы:** тема может быть выбрана из предложенных в рабочей программе или фонде оценочных средств, либо определена самостоятельно в рамках дисциплины (например, «Применение сверточных нейронных сетей для анализа ИК-спектров», «Нейросетевое моделирование кинетики каталитических реакций»). Выбранную тему необходимо согласовать с преподавателем.
2. **Первичная консультация:** обсудите с руководителем:
 - цель и задачи работы;
 - основные аспекты избранной темы;
 - вопросы подбора литературы;

- предварительный план работы.
- 3. **Работа с литературой:**
 - подберите необходимую литературу самостоятельно (учебники, научные статьи, документация библиотек);
 - составьте рабочий вариант плана, выделив основные вопросы темы и параграфы, раскрывающие их содержание;
 - согласуйте список литературы и план с руководителем.
- 4. **Изучение литературы и конспектирование:**
 - внимательно читайте источники, конспектируйте ключевые положения;
 - отражайте основную идею автора и его позицию по исследуемому вопросу;
 - выявляйте проблемы и намечайте задачи для дальнейшего изучения.
- 5. **Написание текста:**
 - систематизируйте и проанализируйте изученный материал;
 - излагайте материал не только описательно, но и с авторской аналитической оценкой;
 - проявляйте собственное видение проблемы и предлагайте пути ее решения.
- 6. **Редактирование и оформление:**
 - предоставьте рабочий вариант текста руководителю на проверку;
 - доработайте текст с учетом замечаний;
 - оформите работу в соответствии с требованиями.

Требования к написанию доклада:

- Написание 1 доклада является обязательным условием выполнения плана СРС по дисциплине.
- Доклад должен быть написан научным языком, с использованием профессиональной терминологии.
- Объем доклада: 15–20 страниц (без учета приложений).

Структура доклада:

Введение (2–3 страницы): обоснование выбора темы, ее актуальность, объект и предмет исследования, цели и задачи.

Основная часть (2–3 раздела, 10–14 страниц):

- обзор литературы и источников по теме;
- характеристика степени разработанности проблемы;
- авторская аналитическая оценка подходов к решению;
- описание методики нейросетевого моделирования (если применимо);
- результаты и их обсуждение.

Заключение (1–2 страницы): краткое изложение достигнутых результатов, выводы, перспективы развития исследования.

Список использованной литературы (не менее 10 источников): оформляется в алфавитном порядке в соответствии с ГОСТ. Рекомендуется включать работы отечественных и зарубежных авторов, статьи за последние 3–5 лет, ссылки на ресурсы сети Интернет.

Приложение (при необходимости): исходный код, дополнительные графики, таблицы данных.

Требования к оформлению:

- текст с одной стороны листа формата А4;
- шрифт Times New Roman, кегль 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу 2,5 см, слева 3 см, справа 1,5 см;
- нумерация страниц, начиная с титульного листа (на титульном номер не ставится);

– доклад должен быть представлен в сброшюрованном виде или в электронном формате (PDF).

Порядок защиты доклада:

– Защита проводится на практических/лабораторных занятиях после исправления всех замечаний руководителя.

– На защиту отводится 5кк7 минут.

– Студент должен показать свободное владение материалом, умение отвечать на вопросы.

– Приветствуется использование мультимедиа-презентации (схемы архитектур, графики результатов).

Критерии оценки доклада:

– соблюдение требований к оформлению;

– полнота и достаточность информации для раскрытия темы;

– умение студента свободно излагать основные идеи, отраженные в докладе;

– способность понять суть вопросов и сформулировать точные ответы;

– наличие авторской позиции и аналитического подхода.

Рейтинговый балл, выставляемый студенту за текущее контрольное мероприятие, сданное студентом в установленные графиком контрольных мероприятий сроки, определяется следующим образом:

Уровень выполнения контрольного задания	Рейтинговый балл (в % от максимального балла за контрольное задание)
Отличный	100
Хороший	80
Удовлетворительный	60
Неудовлетворительный	0

4.5. Методические рекомендации по выполнению исследовательских проектов

Исследовательский проект по дисциплине – это индивидуальная или групповая (2–3 человека) работа, для выполнения которой необходим выбор и применение научной методики к поставленной задаче, получение собственного теоретического или вычислительного результата, на основании которого проводится анализ и формулируются выводы.

Цель проекта: продемонстрировать владение навыками нейросетевого моделирования химико-технологических процессов: от постановки задачи до интерпретации результатов.

Этапы выполнения проекта:

1. **Выбор темы:** тема может быть предложена преподавателем или выбрана студентом самостоятельно в рамках дисциплины (например, «Моделирование выхода продукта каталитического процесса с использованием многослойного перцептрона», «Сравнительный анализ архитектур нейронных сетей для прогнозирования параметров реакции»). Тему необходимо согласовать с преподавателем.
2. **Планирование:**
 - сформулируйте цель и задачи проекта;
 - определите источники данных (синтетическая генерация, открытые датасеты, экспериментальные данные);
 - выберите архитектуру модели и методы оценки качества;
 - составьте календарный план выполнения этапов.
3. **Реализация:**

- подготовьте данные: очистка, кодирование, стандартизация;
- создайте и обучите нейросетевую модель в Google Colab;
- проведите валидацию на тестовой выборке;
- выполните прогноз для новых условий и проанализируйте результаты.

4. Оформление результатов:

- подготовьте отчет по структуре лабораторной работы №9 (сквозной проект);
- визуализируйте результаты: графики обучения, «факт/прогноз», анализ остатков;
- сформулируйте выводы и рекомендации.

5. Подготовка к защите:

- подготовьте презентацию (5–7 слайдов): цель, методика, результаты, выводы;
- отрепетируйте выступление (7–10 минут);
- подготовьте ответы на возможные вопросы.

Требования к оформлению отчета по проекту:

– Структура: титульный лист, введение, основная часть (постановка задачи, методика, результаты), заключение, список литературы, приложение (код, дополнительные материалы). – Объем: 10–15 страниц основного текста. – Оформление: согласно требованиям к лабораторным работам (шрифт, поля, интервалы).

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком уровне, материал фактически верен, студент свободно отвечает на вопросы	Технические средства и ИТ освоены и использованы полностью	Проявлена инициатива, творческий подход, навыки работы в команде (для группового проекта).	Проект представлен полностью и в срок
«Хорошо»	Допущено до 4–5 негрубых ошибок, ответы на вопросы достаточно полные	Обнаруживаются отдельные ошибки в использовании ИТ	Задачи выполнены без инициативы, но полно	Проект представлен с незначительными недоработками
«Удовлетворительно»	Допущено до 8 ошибок, ответы на вопросы фрагментарны	Недостаточное владение навыками работы с ИТ	Выполнена большая часть работы	Проект сдан с опозданием или не полностью

«Неудовлетворительно»	Работа выполнена не или содержит существенные ошибки, студент не ориентируется в материале	Навыки работы с ИТ не продемонстрированы	Студент практически не участвовал в выполнении проекта	Проект не сдан
-----------------------	--	--	--	----------------

- Студенты защищают проект в режиме презентации (очно или онлайн).
- Предъявляются файлы выполненного проекта (ноутбук Colab, отчет в PDF).
- Студент должен уметь рассказать о технологиях, использованных при выполнении, и дать оценку вклада каждого участника (для группового проекта).

4.6. Методические рекомендации по подготовке к экзаменам и зачетам

Общий результат оценивания знаний и умений для выставления зачета может формироваться как интегральная оценка, складывающаяся из текущего контроля (например, выполнение заданий на лабораторно-практических занятиях, защиты индивидуальных творческих работ) и промежуточного контроля (презентация доклада, собеседование, тестирование).

Зачет может проводиться в форме тестирования и просмотра представленных творческих работ. На тестирование обычно отводится ограниченное время (например, 15–20 минут).

Критерии оценивания знаний и умений для выставления зачета:

- Оценка «Отлично»: Студент показал высокий уровень сформированности компетенций: всесторонние, систематизированные, глубокие знания дисциплины; умеет уверенно применять их на практике при решении конкретных задач; свободно и правильно обосновывает принятые решения; демонстрирует творческий подход и аналитическое мышление.
- «Хорошо» Студент показал уровень сформированности компетенций выше среднего: твердо знает материал, грамотно и по существу излагает его, умеет применять знания на практике, но допускает не критичные неточности в ответах или решении задач.
- «Удовлетворительно» Студент показал уровень сформированности компетенций ниже среднего: знания носят фрагментарный характер, допускаются недостаточно правильные формулировки, нарушения логической последовательности. При этом студент владеет основными разделами программы, необходимыми для дальнейшего обучения, и может применять знания по образцу в стандартной ситуации.
- «Неудовлетворительно» Студент не усвоил значительную часть программного материала, ответ содержит существенные ошибки, не способен применить знания даже в стандартной ситуации.

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка доклада, оценка презентации, оценка выполнения проекта и/или индивидуальных творческих работ.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Основная литература:

1. Павлова, А. И. Искусственные нейронные сети: учебное пособие / А. И. Павлова. – Москва: Ай Пи Ар Медиа, 2021. – 190 с. – ISBN 978-5-4497-1165-6. – Текст: электронный // ЭБС IPRbooks.
2. Барский, А. Б. Введение в нейронные сети: учебное пособие / А. Б. Барский. – 4-е изд. – Москва: ИНТУИТ, Ай Пи Ар Медиа, 2024. – 357 с. – ISBN 978-5-4497-2381-9. – Текст: электронный // ЭБС IPRbooks.
3. Иванюк, В. А. Практикум по нейронным сетям: учебное пособие / В. А. Иванюк. – Москва: Прометей, 2024. – 230 с. – ISBN 978-5-00172-601-2. – Текст: электронный // ЭБС «Университетская библиотека онлайн».

Дополнительная литература:

1. Яхьяева, Г. Э. Нечеткие множества и нейронные сети: учебное пособие / Г. Э. Яхьяева. – 5-е изд. – Москва: ИНТУИТ, Ай Пи Ар Медиа, 2024. – 315 с. – ISBN 978-5-4497-3309-2. – Текст: электронный // ЭБС IPRbooks.
2. Вакуленко, С. А. Нейронные сети: учебное пособие / С. А. Вакуленко, А. А. Жихарева. – Санкт-Петербург: СПбГУПТД, 2019. – 110 с. – Текст: электронный // ЭБС «Университетская библиотека онлайн».
3. Goodfellow, I. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – MIT Press, 2016. – 800 p. – URL: <https://www.deeplearningbook.org> (англ.).
4. Chollet, F. Deep Learning with Python / F. Chollet. – Manning Publications, 2021. – 504 p. – URL: <https://www.manning.com/books/deep-learning-with-python-second-edition> (англ.).

Методическая литература:

1. Методические указания к выполнению лабораторных работ по дисциплине «Нейросетевое моделирование технологических процессов» / сост. У. А. Ляхова. – Ставрополь: СКФУ, 2026.
2. Методические рекомендации по организации самостоятельной работы студентов химического факультета. – Ставрополь: СКФУ, 2026.

Интернет-ресурсы:

1. Официальная документация TensorFlow: <https://www.tensorflow.org>
2. Документация Keras: <https://keras.io>
3. Документация scikit-learn: <https://scikit-learn.org>
4. Научная электронная библиотека eLIBRARY.RU: <https://elibrary.ru>
5. Платформа для выполнения кода Google Colab: <https://colab.research.google.com>
6. Электронная информационно-образовательная среда СКФУ: <http://el.ncfu.ru>
7. Репозиторий открытых датасетов Kaggle: <https://www.kaggle.com/datasets>
8. <http://el.ncfu.ru/> – система управления обучением ФГАОУ ВО СКФУ.