Python

1. Встроенные типы данных Python. Функция type(). Приведение типов.
2. Условный оператор.
3. Арифметические и логические операции.
4. Коллекции данных: типы tuple, list, dict,set.
5. Переменные Python. Правила именования переменных.
6. Циклы for и while.
7. Функции ввода/вывода данных Python (print(), input()).
8. Пользовательские функции. Параметры функций.
9. Библиотека Numpy. Индексация. Срезы.
10. Библиотека Numpy. Модуль random.
11. Библиотека Matplotlib. Методы .plot(), .bar(), .hist().
12. Модули. Модуль os.
13. Работа с файлами. Функция open(). Контекстный менеджер with.
14. Библиотека Pandas. Базовые методы.
15. Библиотека Pandas. Статистические методы.
16. REST API. Основные методы: GET, POST
17. Fast API. Создание и запуск приложения с помощью данного фреймворка

Нейронные сети

1. Активационные функции.
2. Функция ошибки. Стандартные функции ошибок keras.
3. Оптимизаторы. Стандартные оптимизаторы keras.
4. Обучение нейронной сети. Выборки. Валидация данных.
5. Задача классификации. Формирование выборки, особенности построения архитектур НС.
6. Переобучение нейронной сети. Причины. Способы борьбы.
7. Особенности построения моделей с помощью Sequential и FunctionalAPI.
8. Линейный слой Dense. Устройство слоя.
9. Сверточный слой Conv2D. Основные параметры слоя.
10. Pooling-слои.
11. Tokenizer. Применение, основные параметры.
12. Параметризация текста BagOfWords.
13. Embedding-слой. Основные параметры слоя.
14. Задач регрессии. Примеры. Подготовка данных, принцип построения архитектур НС для решения задачи.
15. Временный ряды. Инструмент TimeSeriesGenerator. Особенности формирования выборки.
16. Аугментация изображений. Инструмент ImageDataGenerator.
17. Устройство автокодировщика. Области применения и недостатки подхода.
18. Способы сохранения/загрузки модели нейронной сети.
19. Архитектура Вариационных автокодировщиков. Ошибка Кульбака-Лейблера.
20. Инструмент Callback’ов. Встроенные Callback’и keras’а.
21. Архитектура U-Net. Принцип построения.
22. Задача сегментации изображений. Формирование выборки.
23. Задача кластеризации. Алгоритм Kmean.
24. Рекуррентные нейронные сети. Слой LSTM.
25. Устройство GAN.
26. GAN с условием.
27. Основные принципе алгоритма Q-learning.
28. Основные принципе алгоритма Police Gradient.
29. Архитектура SequenceToSequence. Принцип построения.
30. Сети с вниманием. Основные преимущества по сравнению с моделью seq2seq.
31. Обнаружение объектов. Принцип построения архитектуры YOLO.
32. Параметризация аудио. Основные фичи, извлекаемые из аудиофайла.
33. Базовые принципы генетического алгоритма.
34. Подбор гиперпараметров нейронной сети с помощью генетического алгоритма.
35. Распознавание речи. Метрика WER.
36. Синтез речи. Конкатенативный и параметрический подходы. Библиотеки синтеза речи.

Вопрос 1. Встроенные типы данных Python. Функция type(). Приведение типов.

### \*\*Экзаменационный вопрос: Встроенные типы данных Python. Функция `type()`. Приведение типов.\*\*

#### \*\*1. Встроенные типы данных в Python\*\*

Python поддерживает следующие \*\*основные встроенные (примитивные) типы данных\*\*:

1. \*\*Числовые типы\*\*:

- `int` — целые числа (например, `5`, `-3`, `42`).

- `float` — числа с плавающей запятой (например, `3.14`, `-0.001`).

- `complex` — комплексные числа (например, `1 + 2j`).

2. \*\*Логический тип\*\*:

- `bool` — булевы значения (`True` или `False`).

3. \*\*Последовательности\*\*:

- `str` — строки (например, `"Hello"`, `'Python'`).

- `list` — списки (изменяемые, например, `[1, 2, 3]`).

- `tuple` — кортежи (неизменяемые, например, `(1, 2, 3)`).

- `range` — диапазон (например, `range(5)`).

4. \*\*Множества\*\*:

- `set` — изменяемое множество (например, `{1, 2, 3}`).

- `frozenset` — неизменяемое множество.

5. \*\*Словари\*\*:

- `dict` — словарь (пары ключ-значение, например, `{"name": "Alice", "age": 25}`).

6. \*\*Бинарные типы\*\*:

- `bytes` — неизменяемая последовательность байт (например, `b'hello'`).

- `bytearray` — изменяемая последовательность байт.

- `memoryview` — доступ к памяти объекта.

7. \*\*None\*\*:

- `NoneType` — отсутствие значения (например, `None`).

---

#### \*\*2. Функция `type()`\*\*

Функция `type()` возвращает тип переданного объекта:

```python

print(type(10)) # <class 'int'>

print(type(3.14)) # <class 'float'>

print(type("Hello")) # <class 'str'>

print(type([1, 2, 3])) # <class 'list'>

print(type(True)) # <class 'bool'>

print(type(None)) # <class 'NoneType'>

```

\*\*Применение\*\*:

- Проверка типа переменной.

- Отладка кода.

---

#### \*\*3. Приведение типов (Type Casting)\*\*

Приведение типов — преобразование одного типа данных в другой с помощью встроенных функций:

1. \*\*Явное приведение\*\*:

- `int(x)` — преобразует `x` в целое число.

```python

print(int("123")) # 123 (str → int)

print(int(3.99)) # 3 (float → int, округляет вниз)

```

- `float(x)` — преобразует `x` в число с плавающей запятой.

```python

print(float("3.14")) # 3.14 (str → float)

```

- `str(x)` — преобразует `x` в строку.

```python

print(str(100)) # "100" (int → str)

```

- `bool(x)` — преобразует `x` в булево значение.

```python

print(bool(0)) # False (int → bool)

print(bool("text")) # True (str → bool)

```

2. \*\*Неявное приведение\*\*:

Python автоматически приводит типы в арифметических операциях:

```python

print(10 + 3.5) # 13.5 (int + float → float)

```

\*\*Ошибки приведения\*\*:

```python

int("abc") # ValueError (нельзя преобразовать строку с буквами в int)

float(None) # TypeError

```

---

### \*\*Итог\*\*

- Python имеет богатый набор встроенных типов данных.

- `type()` помогает определить тип объекта.

- Приведение типов выполняется функциями `int()`, `float()`, `str()`, `bool()` и другими.

- Важно учитывать возможные ошибки при несовместимости типов.

Пример на закрепление:

```python

a = "5"

b = 2

print(int(a) + b) # 7 (str "5" → int 5)

```

Вопрос 2: Условный оператор

### \*\*Экзаменационный вопрос: Условный оператор в Python\*\*

Условный оператор позволяет выполнять различные действия в зависимости от выполнения определенных условий. В Python реализован через конструкции \*\*`if-elif-else`\*\*.

---

## \*\*1. Базовый синтаксис условного оператора\*\*

### \*\*1.1 Оператор `if` (если)\*\*

Проверяет условие и выполняет блок кода, если оно истинно (`True`).

```python

if условие:

# выполняется, если условие True

действие

```

\*\*Пример:\*\*

```python

x = 10

if x > 5:

print("x больше 5") # Выведется, так как 10 > 5

```

---

### \*\*1.2 Оператор `else` (иначе)\*\*

Выполняется, если условие в `if` ложно (`False`).

```python

if условие:

действие1 # если True

else:

действие2 # если False

```

\*\*Пример:\*\*

```python

age = 17

if age >= 18:

print("Доступ разрешён")

else:

print("Доступ запрещён") # Выведется, так как 17 < 18

```

---

### \*\*1.3 Оператор `elif` (else if — "иначе если")\*\*

Позволяет проверять несколько условий подряд.

```python

if условие1:

действие1

elif условие2:

действие2

else:

действие3

```

\*\*Пример:\*\*

```python

score = 85

if score >= 90:

print("Отлично!")

elif score >= 70:

print("Хорошо!") # Выведется, так как 85 >= 70

else:

print("Плохо :(")

```

---

## \*\*2. Вложенные условия\*\*

Условные операторы можно вкладывать друг в друга.

```python

if условие1:

if условие2:

действие

else:

другое\_действие

else:

иное\_действие

```

\*\*Пример:\*\*

```python

num = 15

if num > 10:

if num % 2 == 0:

print("Число больше 10 и чётное")

else:

print("Число больше 10, но нечётное") # Выведется

else:

print("Число 10 или меньше")

```

---

## \*\*3. Тернарный оператор (сокращённая запись `if-else`)\*\*

Используется для простых условий в одну строку.

\*\*Синтаксис:\*\*

```python

переменная = значение\_если\_True if условие else значение\_если\_False

```

\*\*Пример:\*\*

```python

x = 10

result = "Чётное" if x % 2 == 0 else "Нечётное"

print(result) # Выведет "Чётное"

```

---

## \*\*4. Логические операторы в условиях\*\*

- `and` (И) — оба условия должны быть `True`.

- `or` (ИЛИ) — хотя бы одно условие должно быть `True`.

- `not` (НЕ) — инвертирует условие.

\*\*Пример:\*\*

```python

age = 25

has\_ticket = True

if age >= 18 and has\_ticket:

print("Вход разрешён") # Выведется

else:

print("Вход запрещён")

```

---

## \*\*5. Проверка на `None` и пустые коллекции\*\*

- `None` — отсутствие значения.

- Пустые коллекции (`[]`, `""`, `{}`, `set()`) считаются `False`.

\*\*Пример:\*\*

```python

name = ""

if name: # эквивалентно if len(name) > 0

print(f"Привет, {name}!")

else:

print("Имя не указано") # Выведется

```

---

## \*\*Вывод\*\*

- Условный оператор \*\*`if-elif-else`\*\* позволяет управлять потоком выполнения программы.

- Можно использовать \*\*вложенные условия\*\* и \*\*тернарный оператор\*\* для краткости.

- \*\*Логические операторы\*\* (`and`, `or`, `not`) помогают комбинировать условия.

- Пустые коллекции и `None` интерпретируются как `False`.

\*\*Пример итогового кода:\*\*

```python

weather = "дождь"

if weather == "солнце":

print("Идём гулять!")

elif weather == "дождь":

print("Берём зонт") # Выведется

else:

print("Остаёмся дома")

```

Вопрос №3. Арифметические и логические операции

### \*\*Экзаменационный вопрос: Арифметические и логические операции в Python\*\*

---

## \*\*1. Арифметические операции\*\*

Арифметические операции выполняют математические вычисления над числами (`int`, `float`, `complex`).

### \*\*Основные арифметические операторы:\*\*

| Оператор | Описание | Пример | Результат |

|----------|-------------------|--------------|-----------|

| `+` | Сложение | `5 + 3` | `8` |

| `-` | Вычитание | `10 - 2` | `8` |

| `\*` | Умножение | `4 \* 3` | `12` |

| `/` | Деление | `10 / 2` | `5.0` |

| `//` | Целочисленное деление | `10 // 3` | `3` |

| `%` | Остаток от деления | `10 % 3` | `1` |

| `\*\*` | Возведение в степень | `2 \*\* 3` | `8` |

### \*\*Примеры:\*\*

```python

a = 10

b = 3

print(a + b) # 13

print(a / b) # 3.333... (возвращает float)

print(a // b) # 3 (целочисленное деление)

print(a % b) # 1 (остаток)

print(a \*\* b) # 1000 (10³)

```

### \*\*Специальные случаи:\*\*

- Деление на ноль вызывает ошибку: `ZeroDivisionError`.

- Операции между `int` и `float` возвращают `float`:

```python

print(5 + 2.0) # 7.0 (float)

```

---

## \*\*2. Логические операции\*\*

Логические операции работают с булевыми значениями (`True`, `False`) и используются в условиях.

### \*\*Основные логические операторы:\*\*

| Оператор | Описание | Пример | Результат |

|----------|-------------------|---------------------|-----------|

| `and` | Логическое И | `True and False` | `False` |

| `or` | Логическое ИЛИ | `True or False` | `True` |

| `not` | Логическое НЕ | `not True` | `False` |

### \*\*Таблицы истинности:\*\*

#### \*\*`and` (И)\*\*

| A | B | A and B |

|---------|---------|---------|

| `True` | `True` | `True` |

| `True` | `False` | `False` |

| `False` | `True` | `False` |

| `False` | `False` | `False` |

#### \*\*`or` (ИЛИ)\*\*

| A | B | A or B |

|---------|---------|--------|

| `True` | `True` | `True` |

| `True` | `False` | `True` |

| `False` | `True` | `True` |

| `False` | `False` | `False` |

#### \*\*`not` (НЕ)\*\*

| A | not A |

|---------|---------|

| `True` | `False` |

| `False` | `True` |

### \*\*Примеры:\*\*

```python

x = 5

y = 10

print(x < 10 and y > 5) # True (оба условия True)

print(x == 5 or y == 0) # True (первое условие True)

print(not x > 0) # False (x > 0 = True, not True = False)

```

---

## \*\*3. Операторы сравнения\*\*

Используются для сравнения значений и возвращают `True` или `False`.

| Оператор | Описание | Пример | Результат |

|----------|-------------------|--------------|-----------|

| `==` | Равно | `5 == 5` | `True` |

| `!=` | Не равно | `5 != 3` | `True` |

| `>` | Больше | `10 > 5` | `True` |

| `<` | Меньше | `3 < 2` | `False` |

| `>=` | Больше или равно | `5 >= 5` | `True` |

| `<=` | Меньше или равно | `4 <= 3` | `False` |

### \*\*Пример:\*\*

```python

a = 7

b = 3

print(a == b) # False

print(a != b) # True

print(a >= b) # True

```

---

## \*\*4. Приоритет операций\*\*

Операторы выполняются в определённом порядке (от высшего к низшему приоритету):

1. `()` (скобки)

2. `\*\*` (степень)

3. `\*`, `/`, `//`, `%`

4. `+`, `-`

5. `==`, `!=`, `>`, `<`, `>=`, `<=`

6. `not`

7. `and`

8. `or`

### \*\*Пример:\*\*

```python

result = 5 + 3 \* 2 \*\* 2 # 5 + (3 \* (2²)) = 5 + 12 = 17

print(result) # 17

```

---

## \*\*5. Применение в условиях\*\*

Логические и арифметические операции часто используются вместе в условных конструкциях (`if-else`).

### \*\*Пример:\*\*

```python

age = 18

has\_license = True

if age >= 18 and has\_license:

print("Можно водить машину") # Выведется

else:

print("Нельзя водить")

```

---

## \*\*Вывод\*\*

- \*\*Арифметические операции\*\* (`+`, `-`, `\*`, `/`, `//`, `%`, `\*\*`) выполняют математические вычисления.

- \*\*Логические операции\*\* (`and`, `or`, `not`) работают с булевыми значениями и используются в условиях.

- \*\*Операторы сравнения\*\* (`==`, `!=`, `>`, `<`, `>=`, `<=`) возвращают `True` или `False`.

- \*\*Приоритет операций\*\* определяет порядок вычислений.

\*\*Итоговый пример:\*\*

```python

a = 10

b = 5

c = 7

print((a > b) and (b < c)) # True (10 > 5 И 5 < 7)

print(2 \*\* 3 + 5) # 13 (8 + 5)

```

Вопрос №4 Коллекции данных: типы tuple, list, dict,set

### \*\*Экзаменационный вопрос: Коллекции данных в Python (tuple, list, dict, set)\*\*

---

## \*\*1. Кортеж (`tuple`)\*\*

\*\*Характеристики:\*\*

- ⚠️ \*\*Неизменяемый\*\* (immutable) тип данных.

- Элементы хранятся в круглых скобках `()`.

- Поддерживает индексацию и срезы.

\*\*Создание:\*\*

```python

t = (1, 2, 3, "a") # или без скобок: t = 1, 2, 3

```

\*\*Основные операции:\*\*

```python

print(t[0]) # 1 (индексация)

print(t[1:3]) # (2, 3) (срез)

print(len(t)) # 4 (длина кортежа)

```

\*\*Где используется?\*\*

- Для хранения константных данных (например, координат `(x, y)`).

- Как ключи в словарях (т.к. кортежи неизменяемы).

---

## \*\*2. Список (`list`)\*\*

\*\*Характеристики:\*\*

- ✅ \*\*Изменяемый\*\* (mutable) тип данных.

- Элементы хранятся в квадратных скобках `[]`.

- Поддерживает индексацию, срезы и методы для модификации.

\*\*Создание:\*\*

```python

lst = [1, 2, 3, "a"]

```

\*\*Основные операции:\*\*

```python

lst[0] = 10 # [10, 2, 3, "a"] (изменение элемента)

lst.append(4) # [10, 2, 3, "a", 4] (добавление в конец)

lst.remove("a") # [10, 2, 3, 4] (удаление элемента)

print(lst[::-1]) # [4, 3, 2, 10] (реверс списка)

```

\*\*Где используется?\*\*

- Для хранения и динамического изменения данных (например, списка задач).

---

## \*\*3. Словарь (`dict`)\*\*

\*\*Характеристики:\*\*

- ✅ \*\*Изменяемый\*\* тип данных.

- Хранит пары \*\*ключ: значение\*\* в фигурных скобках `{}`.

- Ключи должны быть \*\*уникальными\*\* и \*\*неизменяемыми\*\* (числа, строки, кортежи).

\*\*Создание:\*\*

```python

d = {"name": "Alice", "age": 25}

```

\*\*Основные операции:\*\*

```python

d["age"] = 26 # Изменение значения

d["city"] = "Moscow" # Добавление новой пары

print(d.keys()) # dict\_keys(['name', 'age', 'city'])

print(d.values()) # dict\_values(['Alice', 26, 'Moscow'])

```

\*\*Где используется?\*\*

- Для хранения структурированных данных (например, JSON-подобных объектов).

---

## \*\*4. Множество (`set`)\*\*

\*\*Характеристики:\*\*

- ✅ \*\*Изменяемый\*\* тип данных (есть неизменяемая версия — `frozenset`).

- Хранит \*\*уникальные\*\* элементы в фигурных скобках `{}` (без ключей).

- Поддерживает математические операции (объединение, пересечение).

\*\*Создание:\*\*

```python

s = {1, 2, 3, 3} # {1, 2, 3} (дубли удаляются)

```

\*\*Основные операции:\*\*

```python

s.add(4) # {1, 2, 3, 4}

s.discard(2) # {1, 3, 4}

print(s & {3,4,5}) # {3,4} (пересечение)

```

\*\*Где используется?\*\*

- Для удаления дубликатов из списка: `list(set([1,2,2,3])) → [1,2,3]`.

- Проверка принадлежности элемента (`if x in set`).

---

## \*\*Сравнение коллекций\*\*

| Тип | Изменяемость | Уникальность | Индексация | Пример |

|----------|--------------|--------------|------------|-----------------|

| `tuple` | Нет | Нет | Да | `(1, 2, 3)` |

| `list` | Да | Нет | Да | `[1, 2, 3]` |

| `dict` | Да | Ключи — да | По ключу | `{"a": 1}` |

| `set` | Да | Да | Нет | `{1, 2, 3}` |

---

## \*\*Итог\*\*

- \*\*`tuple`\*\* — для неизменяемых последовательностей.

- \*\*`list`\*\* — для динамических коллекций с возможностью изменения.

- \*\*`dict`\*\* — для хранения пар ключ-значение.

- \*\*`set`\*\* — для работы с уникальными элементами.

\*\*Пример преобразований:\*\*

```python

lst = [1, 2, 2, 3]

unique = set(lst) # {1, 2, 3}

back\_to\_list = list(unique) # [1, 2, 3]

```

Вопрос №5 Переменные Python. Правила именования переменных

### \*\*Экзаменационный вопрос: Переменные в Python. Правила именования\*\*

---

## \*\*1. Что такое переменная?\*\*

\*\*Переменная\*\* — это именованная область памяти, которая хранит данные определенного типа. В Python переменные создаются в момент присваивания значения.

```python

x = 10 # Целое число (int)

name = "Alice" # Строка (str)

is\_valid = True # Булево значение (bool)

```

---

## \*\*2. Динамическая типизация\*\*

Python — язык с \*\*динамической типизацией\*\*: тип переменной определяется автоматически при присваивании значения и может меняться.

```python

var = 42 # var имеет тип int

var = "Python" # Теперь var имеет тип str

```

---

## \*\*3. Правила именования переменных\*\*

### \*\*3.1 Разрешено\*\*

- Имена могут содержать:

- Латинские буквы (`a-z`, `A-Z`).

- Цифры (`0-9`), \*\*но не с первого символа\*\*.

- Нижнее подчеркивание `\_`.

- Регистр букв имеет значение (`age ≠ Age ≠ AGE`).

\*\*Примеры допустимых имен:\*\*

```python

user\_name = "John" # с подчеркиванием

count1 = 10 # цифры не в начале

PI = 3.14 # константы (стиль, а не правило)

```

### \*\*3.2 Запрещено\*\*

- Использовать \*\*ключевые слова\*\* Python (например, `if`, `for`, `while`).

- Начинать имя с цифры (`1var` — ошибка).

- Использовать спецсимволы (`@`, `#`, `$` и др.).

\*\*Примеры недопустимых имен:\*\*

```python

1st\_place = "Ivan" # SyntaxError (начинается с цифры)

class = "A" # SyntaxError (ключевое слово)

user@name = "Alex" # SyntaxError (спецсимвол)

```

---

## \*\*4. Стили именования\*\*

- \*\*snake\_case\*\* (для переменных и функций):

```python

user\_age = 25

max\_count = 100

```

- \*\*UPPER\_CASE\*\* (для констант):

```python

PI = 3.14159

MAX\_USERS = 1000

```

- \*\*CamelCase\*\* (для классов):

```python

class UserProfile: ...

```

---

## \*\*5. Особенности Python\*\*

- Переменные \*\*не требуют объявления типа\*\*.

- Нет ограничений на длину имени (но разумная кратность приветствуется).

- Поддержка \*\*множественного присваивания\*\*:

```python

x, y, z = 1, 2, 3 # x=1, y=2, z=3

a = b = c = 0 # a=0, b=0, c=0

```

---

## \*\*6. Проверка типа переменной\*\*

Функция `type()` возвращает тип объекта:

```python

print(type(10)) # <class 'int'>

print(type("Hello")) # <class 'str'>

```

---

## \*\*7. Удаление переменной\*\*

Оператор `del` удаляет переменную:

```python

x = 10

del x

print(x) # NameError: name 'x' is not defined

```

---

## \*\*Итог\*\*

- Переменные создаются при присваивании (`x = 5`).

- Имена должны:

- Начинаться с буквы или `\_`.

- Не содержать спецсимволов и ключевых слов.

- Стили именования: `snake\_case`, `UPPER\_CASE`, `CamelCase`.

- Python сам определяет тип переменной.

\*\*Пример на закрепление:\*\*

```python

first\_name = "Anna" # Правильно

\_last\_name = "Smith" # Правильно (но обычно означает "private")

# 2nd\_place = "Bob" # Ошибка (начинается с цифры)

```

Вопрос №6 Циклы for и while

### \*\*Экзаменационный вопрос: Циклы `for` и `while` в Python\*\*

---

## \*\*1. Цикл `for`\*\*

\*\*Назначение:\*\*

Используется для \*\*итерации по последовательностям\*\* (спискам, строкам, словарям и т.д.) или выполнения блока кода \*\*заданное число раз\*\*.

### \*\*Синтаксис:\*\*

```python

for переменная in последовательность:

тело\_цикла

```

### \*\*Примеры:\*\*

1. \*\*Итерация по списку:\*\*

```python

fruits = ["apple", "banana", "cherry"]

for fruit in fruits:

print(fruit)

```

Вывод:

```

apple

banana

cherry

```

2. \*\*Итерация по строке:\*\*

```python

for char in "Python":

print(char)

```

3. \*\*Использование `range()`:\*\*

```python

for i in range(5): # 0, 1, 2, 3, 4

print(i)

```

4. \*\*Цикл с `else` (выполняется после завершения цикла):\*\*

```python

for i in range(3):

print(i)

else:

print("Цикл завершен")

```

---

## \*\*2. Цикл `while`\*\*

\*\*Назначение:\*\*

Выполняет блок кода \*\*пока условие истинно\*\* (`True`).

### \*\*Синтаксис:\*\*

```python

while условие:

тело\_цикла

```

### \*\*Примеры:\*\*

1. \*\*Простой счетчик:\*\*

```python

count = 0

while count < 5:

print(count)

count += 1

```

Вывод:

```

0

1

2

3

4

```

2. \*\*Бесконечный цикл с `break`:\*\*

```python

while True:

user\_input = input("Введите 'stop' для выхода: ")

if user\_input == "stop":

break # выход из цикла

```

3. \*\*Цикл с `else` (если не было `break`):\*\*

```python

num = 5

while num > 0:

print(num)

num -= 1

else:

print("Цикл завершен без break")

```

---

## \*\*3. Ключевые слова для управления циклами\*\*

- `break` — досрочный выход из цикла.

- `continue` — переход к следующей итерации, пропуская оставшийся код.

- `pass` — заглушка (пустой оператор).

### \*\*Пример:\*\*

```python

for i in range(10):

if i == 3:

continue # пропускаем 3

if i == 8:

break # выходим на 8

print(i)

```

Вывод:

```

0

1

2

4

5

6

7

```

---

## \*\*4. Различия между `for` и `while`\*\*

| Критерий | `for` | `while` |

|----------------|--------------------------------|------------------------------|

| \*\*Использование\*\* | Для итерации по последовательностям. | Пока условие истинно. |

| \*\*Условие\*\* | Не требует явного условия. | Требует условия. |

| \*\*Риск\*\* | Нет риска бесконечного цикла. | Может быть бесконечным. |

| \*\*Пример\*\* | `for i in range(5): ...` | `while x < 10: ...` |

---

## \*\*5. Важные замечания\*\*

- \*\*Бесконечные циклы:\*\*

Убедитесь, что условие в `while` рано или поздно станет `False`, иначе программа зависнет.

- \*\*Оптимизация:\*\*

Для перебора элементов списка/строки предпочтительнее `for`.

- \*\*Вложенные циклы:\*\*

Циклы можно вкладывать друг в друга.

### \*\*Пример вложенных циклов:\*\*

```python

for i in range(3):

for j in range(2):

print(f"i={i}, j={j}")

```

Вывод:

```

i=0, j=0

i=0, j=1

i=1, j=0

i=1, j=1

i=2, j=0

i=2, j=1

```

---

## \*\*Итог\*\*

- \*\*`for`\*\* — для итерации по коллекциям или заданному диапазону.

- \*\*`while`\*\* — для выполнения кода, пока условие истинно.

- Управляйте циклами с помощью `break`, `continue`, `pass`.

- Избегайте бесконечных циклов в `while`.

\*\*Финальный пример:\*\*

```python

# Вывод четных чисел от 0 до 10

for num in range(11):

if num % 2 != 0:

continue

print(num)

```

Вопрос №7 Функции ввода/вывода данных Python (print(), input())

### \*\*Экзаменационный вопрос: Функции ввода/вывода данных в Python (`print()`, `input()`)\*\*

---

## \*\*1. Функция `print()`\*\*

\*\*Назначение:\*\*

Выводит данные на экран (стандартный вывод).

### \*\*Синтаксис:\*\*

```python

print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

```

### \*\*Параметры:\*\*

| Параметр | Описание | Пример |

|----------|--------------------------------------------------------------------------|----------------------------|

| `\*objects` | Объекты для вывода (через запятую). | `print("Hello", "World")` |

| `sep` | Разделитель между объектами (по умолчанию пробел `' '`). | `print(1, 2, sep="-")` → `1-2` |

| `end` | Символ в конце строки (по умолчанию `\n` — перенос строки). | `print("Hello", end="!")` → `Hello!` |

| `file` | Файл для вывода (по умолчанию `sys.stdout` — консоль). | `print("Error", file=sys.stderr)` |

| `flush` | Принудительный сброс буфера вывода (`True`/`False`). | Редко используется. |

### \*\*Примеры:\*\*

1. \*\*Простой вывод:\*\*

```python

print("Hello, World!") # Hello, World!

```

2. \*\*Вывод нескольких значений:\*\*

```python

name = "Alice"

age = 25

print("Name:", name, "Age:", age) # Name: Alice Age: 25

```

3. \*\*Изменение разделителя и окончания:\*\*

```python

print(1, 2, 3, sep=", ", end=".\n") # 1, 2, 3.

```

4. \*\*Форматированный вывод (f-строки):\*\*

```python

print(f"{name} is {age} years old.") # Alice is 25 years old.

```

---

## \*\*2. Функция `input()`\*\*

\*\*Назначение:\*\*

Считывает данные, введенные пользователем с клавиатуры (стандартный ввод).

### \*\*Синтаксис:\*\*

```python

input([prompt]) # prompt — подсказка для пользователя (необязательный параметр).

```

### \*\*Особенности:\*\*

- Возвращает строку (`str`), даже если введено число.

- Для преобразования в другие типы данных используйте функции `int()`, `float()` и т.д.

### \*\*Примеры:\*\*

1. \*\*Простой ввод:\*\*

```python

name = input("Введите ваше имя: ")

print(f"Привет, {name}!")

```

2. \*\*Ввод числа:\*\*

```python

age = int(input("Сколько вам лет? ")) # Преобразование в int

print(f"Через 10 лет вам будет {age + 10}.")

```

3. \*\*Обработка ошибок при вводе:\*\*

```python

try:

number = float(input("Введите число: "))

except ValueError:

print("Ошибка: введено не число!")

```

---

## \*\*3. Комбинация `input()` и `print()`\*\*

```python

city = input("Введите ваш город: ")

print(f"Вы живете в {city}.")

```

---

## \*\*4. Важные замечания\*\*

1. \*\*Тип данных `input()`:\*\*

Всегда возвращает строку. Для чисел требуется явное преобразование:

```python

num = int(input()) # или float(input())

```

2. \*\*Безопасность:\*\*

- `input()` не фильтрует ввод (пользователь может ввести что угодно).

- Всегда проверяйте данные, особенно если они используются в вычислениях.

3. \*\*Перенос строки в `print()`:\*\*

По умолчанию `print()` добавляет `\n` в конец. Чтобы избежать этого:

```python

print("Строка 1", end="")

print("Строка 2") # Строка 1Строка 2

```

---

## \*\*5. Сравнение `print()` и `input()`\*\*

| Критерий | `print()` | `input()` |

|---------------|------------------------------------|------------------------------------|

| \*\*Назначение\*\* | Вывод данных на экран. | Чтение данных с клавиатуры. |

| \*\*Возврат\*\* | Нет (None). | Строка (`str`). |

| \*\*Использование\*\* | `print("Текст")`. | `variable = input("Подсказка")`. |

| \*\*Безопасность\*\* | Безопасен. | Требует проверки ввода. |

---

## \*\*Итог\*\*

- \*\*`print()`\*\* — для вывода данных с гибкими настройками (разделитель, конец строки).

- \*\*`input()`\*\* — для ввода данных (всегда возвращает `str`).

- Всегда проверяйте и преобразовывайте ввод, если нужны числа.

- Используйте f-строки для удобного форматирования вывода.

\*\*Финальный пример:\*\*

```python

# Калькулятор суммы двух чисел

a = int(input("Введите первое число: "))

b = int(input("Введите второе число: "))

print(f"Сумма: {a + b}")

```

Вопрос №8 Пользовательские функции. Параметры функций.

### \*\*Экзаменационный вопрос: Пользовательские функции. Параметры функций\*\*

---

## \*\*1. Пользовательские функции\*\*

\*\*Функция\*\* — это блок кода, который выполняет определенную задачу и может быть вызван по имени.

### \*\*1.1. Создание функции\*\*

\*\*Синтаксис:\*\*

```python

def имя\_функции(параметры):

"""Документация (опционально)"""

тело\_функции

return результат # (опционально)

```

\*\*Пример:\*\*

```python

def greet(name):

"""Функция приветствия пользователя."""

print(f"Привет, {name}!")

greet("Анна") # Вызов функции

```

\*\*Вывод:\*\*

```

Привет, Анна!

```

---

## \*\*2. Параметры функций\*\*

Параметры — это переменные, которые передаются в функцию при ее вызове.

### \*\*2.1. Типы параметров\*\*

#### \*\*1. Позиционные параметры\*\*

Передаются в строгом порядке.

```python

def add(a, b): # a и b — позиционные параметры

return a + b

print(add(3, 5)) # 8

```

#### \*\*2. Именованные параметры\*\*

Указываются с именами при вызове (порядок не важен).

```python

print(add(b=5, a=3)) # 8

```

#### \*\*3. Параметры по умолчанию\*\*

Имеют предустановленные значения.

```python

def greet(name="Гость"): # name — параметр по умолчанию

print(f"Привет, {name}!")

greet() # Привет, Гость!

greet("Алиса") # Привет, Алиса!

```

#### \*\*4. Произвольное число аргументов\*\*

- `\*args` — кортеж позиционных аргументов.

- `\*\*kwargs` — словарь именованных аргументов.

\*\*Пример:\*\*

```python

def sum\_all(\*args):

return sum(args)

print(sum\_all(1, 2, 3)) # 6

def user\_info(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

user\_info(name="Анна", age=25)

```

\*\*Вывод:\*\*

```

name: Анна

age: 25

```

---

## \*\*3. Возврат значений (`return`)\*\*

Функция может возвращать результат с помощью `return`.

### \*\*3.1. Возврат одного значения\*\*

```python

def square(x):

return x \*\* 2

print(square(4)) # 16

```

### \*\*3.2. Возврат нескольких значений (кортеж)\*\*

```python

def divide(a, b):

return a // b, a % b # Возвращает частное и остаток

quotient, remainder = divide(10, 3)

print(quotient, remainder) # 3 1

```

### \*\*3.3. Функция без `return`\*\*

Если `return` отсутствует, функция возвращает `None`.

```python

def no\_return():

print("Hello")

result = no\_return()

print(result) # None

```

---

## \*\*4. Область видимости переменных\*\*

- \*\*Локальные переменные\*\* — существуют только внутри функции.

- \*\*Глобальные переменные\*\* — доступны во всей программе.

\*\*Пример:\*\*

```python

x = 10 # Глобальная переменная

def change\_x():

x = 20 # Локальная переменная

print(x) # 20

change\_x()

print(x) # 10 (глобальная x не изменилась)

```

\*\*Как изменить глобальную переменную:\*\*

```python

def change\_global():

global x

x = 20

change\_global()

print(x) # 20

```

---

## \*\*5. Аннотации типов (Python 3.5+)\*\*

Указание типов параметров и возвращаемого значения для удобства чтения.

```python

def add(a: int, b: int) -> int:

return a + b

```

---

## \*\*Итог\*\*

1. \*\*Функции создаются через `def`.\*\*

2. \*\*Параметры бывают:\*\*

- Позиционные (`a, b`).

- Именованные (`name="Гость"`).

- Произвольные (`\*args`, `\*\*kwargs`).

3. \*\*`return` возвращает результат (или `None`).\*\*

4. \*\*Область видимости:\*\*

- Локальные переменные — внутри функции.

- Глобальные — во всей программе.

\*\*Пример итоговой функции:\*\*

```python

def calculate(\*args, operation="+"):

if operation == "+":

return sum(args)

elif operation == "\*":

result = 1

for num in args:

result \*= num

return result

print(calculate(2, 3, 4, operation="\*")) # 24

```

Вопрос №9 Библиотека Numpy. Индексация. Срезы.

### \*\*Экзаменационный вопрос: Библиотека NumPy. Индексация. Срезы\*\*

---

## \*\*1. Библиотека NumPy\*\*

\*\*NumPy\*\* (Numerical Python) — это библиотека для работы с многомерными массивами и математическими операциями над ними.

### \*\*1.1. Основные особенности\*\*

- \*\*ndarray\*\* — многомерный массив (матрицы, тензоры).

- Высокая производительность (реализация на C).

- Поддержка операций линейной алгебры, статистики и др.

### \*\*1.2. Создание массива\*\*

```python

import numpy as np

arr = np.array([1, 2, 3]) # 1D массив

matrix = np.array([[1, 2], [3, 4]]) # 2D массив (матрица)

```

---

## \*\*2. Индексация в NumPy\*\*

### \*\*2.1. Индексация в одномерном массиве\*\*

Аналогично спискам Python:

```python

arr = np.array([10, 20, 30, 40])

print(arr[0]) # 10 (первый элемент)

print(arr[-1]) # 40 (последний элемент)

```

### \*\*2.2. Индексация в многомерных массивах\*\*

Используется несколько индексов (через запятую):

```python

matrix = np.array([[1, 2, 3], [4, 5, 6]])

print(matrix[0, 1]) # 2 (первая строка, второй столбец)

```

---

## \*\*3. Срезы (Slicing)\*\*

Срезы позволяют выбирать подмассивы.

### \*\*3.1. Синтаксис срезов\*\*

```python

массив[начало:конец:шаг]

```

### \*\*3.2. Примеры для 1D массива\*\*

```python

arr = np.array([0, 1, 2, 3, 4, 5])

print(arr[1:4]) # [1, 2, 3] (элементы с 1 по 3)

print(arr[::2]) # [0, 2, 4] (каждый второй)

print(arr[::-1]) # [5, 4, 3, 2, 1, 0] (реверс)

```

### \*\*3.3. Примеры для 2D массива\*\*

```python

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Первые две строки, все столбцы

print(matrix[:2, :]) # [[1, 2, 3], [4, 5, 6]]

# Вторая строка, последние два элемента

print(matrix[1, -2:]) # [5, 6]

```

### \*\*3.4. Важные особенности\*\*

- Срезы возвращают \*\*view\*\* (представление), а не копию. Изменения в `view` влияют на исходный массив.

- Чтобы создать копию, используйте `.copy()`:

```python

sub\_arr = arr[1:4].copy()

```

---

## \*\*4. Булева индексация\*\*

Выбор элементов по условию:

```python

arr = np.array([1, 2, 3, 4, 5])

print(arr[arr > 3]) # [4, 5] (элементы > 3)

```

---

## \*\*5. Fancy-индексация\*\*

Выбор элементов по списку индексов:

```python

arr = np.array([10, 20, 30, 40])

indices = [0, 2]

print(arr[indices]) # [10, 30]

```

---

## \*\*6. Практические примеры\*\*

### \*\*6.1. Замена элементов по условию\*\*

```python

arr = np.array([1, 2, 3, 4])

arr[arr % 2 == 0] = -1 # Четные -> -1

print(arr) # [1, -1, 3, -1]

```

### \*\*6.2. Выбор строк/столбцов в матрице\*\*

```python

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Первый и последний столбец

print(matrix[:, [0, -1]]) # [[1, 3], [4, 6], [7, 9]]

```

---

## \*\*Итог\*\*

1. \*\*NumPy\*\* — библиотека для работы с массивами.

2. \*\*Индексация\*\*:

- Одномерные массивы: `arr[индекс]`.

- Многомерные: `matrix[строка, столбец]`.

3. \*\*Срезы\*\*:

- `arr[start:stop:step]`.

- Возвращают `view` (используйте `.copy()` для копии).

4. \*\*Булева индексация\*\* — выбор по условию.

5. \*\*Fancy-индексация\*\* — выбор по списку индексов.

\*\*Пример на закрепление:\*\*

```python

arr = np.array([5, 10, 15, 20])

print(arr[1:3]) # [10, 15]

print(arr[arr > 10]) # [15, 20]

```

Вопрос №10 Библиотека Numpy. Модуль random.

### \*\*Экзаменационный вопрос: Библиотека NumPy. Модуль random\*\*

---

## \*\*1. Модуль `numpy.random`\*\*

Модуль `numpy.random` предоставляет функции для генерации псевдослучайных чисел и работы с вероятностными распределениями.

### \*\*1.1. Основные возможности\*\*

- Генерация случайных чисел (целых, вещественных).

- Работа с распределениями (нормальное, равномерное и др.).

- Перемешивание массивов.

- Установка случайного seed (для воспроизводимости).

---

## \*\*2. Генерация случайных чисел\*\*

### \*\*2.1. Равномерное распределение\*\*

#### \*\*`rand()`\*\*

Генерирует числа в диапазоне `[0, 1)` с равномерным распределением.

```python

import numpy as np

# Массив 3x2 из случайных чисел [0, 1)

arr = np.random.rand(3, 2)

print(arr)

```

\*\*Вывод:\*\*

```

[[0.123, 0.456],

[0.789, 0.012],

[0.345, 0.678]]

```

#### \*\*`randint()`\*\*

Генерирует \*\*целые\*\* числа в заданном диапазоне.

```python

# 5 случайных целых чисел от 1 до 10

numbers = np.random.randint(1, 10, size=5)

print(numbers) # [3, 7, 2, 9, 4]

```

---

### \*\*2.2. Нормальное распределение\*\*

#### \*\*`randn()`\*\*

Генерирует числа из стандартного нормального распределения (μ=0, σ=1).

```python

# Массив 2x2 из N(0, 1)

arr = np.random.randn(2, 2)

print(arr)

```

\*\*Вывод:\*\*

```

[[-0.5, 1.2],

[0.3, -1.8]]

```

#### \*\*`normal()`\*\*

Генерирует числа из нормального распределения с заданными параметрами (μ, σ).

```python

# 5 чисел с μ=10, σ=2

numbers = np.random.normal(10, 2, 5)

print(numbers) # [9.1, 12.3, 8.5, 10.7, 11.2]

```

---

## \*\*3. Работа с массивами\*\*

### \*\*3.1. Перемешивание (`shuffle`, `permutation`)\*\*

#### \*\*`shuffle()`\*\*

Перемешивает массив \*\*на месте\*\*.

```python

arr = np.array([1, 2, 3, 4])

np.random.shuffle(arr)

print(arr) # [3, 1, 4, 2] (порядок изменён)

```

#### \*\*`permutation()`\*\*

Возвращает перемешанную \*\*копию\*\* массива.

```python

arr = np.array([1, 2, 3])

shuffled = np.random.permutation(arr)

print(shuffled) # [2, 1, 3] (исходный массив не изменён)

```

---

### \*\*3.2. Выбор случайных элементов (`choice`)\*\*

Функция `choice()` выбирает элементы из массива с заданными вероятностями.

#### \*\*Без возвращения (без повторов)\*\*

```python

# Выбор 3 уникальных элементов из [0, 1, 2, 3, 4]

choices = np.random.choice(5, size=3, replace=False)

print(choices) # [2, 0, 4]

```

#### \*\*С возвращением (возможны повторы)\*\*

```python

choices = np.random.choice(5, size=3, replace=True)

print(choices) # [1, 1, 3]

```

#### \*\*С весами\*\*

```python

# Вероятности: [0.1, 0.1, 0.8]

choices = np.random.choice([1, 2, 3], size=5, p=[0.1, 0.1, 0.8])

print(choices) # [3, 3, 2, 3, 3]

```

---

## \*\*4. Управление случайностью (`seed`)\*\*

Для воспроизводимости результатов можно установить \*\*seed\*\*.

```python

np.random.seed(42) # Фиксируем seed

a = np.random.rand(3)

print(a) # [0.374, 0.950, 0.731] (всегда одинаково)

```

---

## \*\*5. Другие распределения\*\*

| Функция | Распределение | Пример |

|-----------------------|------------------------|----------------------------|

| `exponential()` | Экспоненциальное | `np.random.exponential(1)` |

| `poisson()` | Пуассона | `np.random.poisson(5)` |

| `binomial()` | Биномиальное | `np.random.binomial(10, 0.5)` |

| `uniform()` | Равномерное | `np.random.uniform(0, 10)` |

---

## \*\*6. Примеры использования\*\*

### \*\*6.1. Генерация случайной матрицы\*\*

```python

matrix = np.random.randint(0, 100, size=(3, 3))

print(matrix)

```

\*\*Вывод:\*\*

```

[[12, 45, 67],

[89, 23, 56],

[34, 78, 90]]

```

### \*\*6.2. Моделирование броска кубика\*\*

```python

dice\_roll = np.random.randint(1, 7, size=10)

print(dice\_roll) # [4, 6, 1, 3, 5, 2, 6, 4, 1, 2]

```

---

## \*\*Итог\*\*

1. \*\*`numpy.random`\*\* — модуль для генерации случайных чисел.

2. \*\*Основные функции:\*\*

- `rand()`, `randint()` — равномерное распределение.

- `randn()`, `normal()` — нормальное распределение.

- `shuffle()`, `permutation()` — перемешивание.

- `choice()` — выбор случайных элементов.

3. \*\*Seed\*\* — для воспроизводимости результатов.

4. \*\*Применение:\*\* моделирование, машинное обучение, статистика.

\*\*Финальный пример:\*\*

```python

np.random.seed(123)

data = np.random.normal(0, 1, 100) # 100 чисел из N(0, 1)

print(data[:5]) # [-1.085, 0.997, 0.283, -1.506, -0.579]

```

Вопрос №11 Библиотека Matplotlib. Методы .plot(), .bar(), .hist()

### \*\*Экзаменационный вопрос: Библиотека Matplotlib. Методы `.plot()`, `.bar()`, `.hist()`\*\*

---

## \*\*1. Библиотека Matplotlib\*\*

\*\*Matplotlib\*\* — это основная библиотека Python для визуализации данных. Она позволяет создавать:

- Линейные графики (`plot`),

- Столбчатые диаграммы (`bar`),

- Гистограммы (`hist`),

- И многие другие типы графиков.

\*\*Импорт библиотеки:\*\*

```python

import matplotlib.pyplot as plt

```

---

## \*\*2. Метод `.plot()` — линейные графики\*\*

Используется для построения \*\*линейных графиков\*\* (тренды, временные ряды).

### \*\*2.1. Синтаксис\*\*

```python

plt.plot(x, y, fmt, \*\*kwargs)

```

- `x`, `y` — данные для осей X и Y,

- `fmt` — строка формата (цвет, стиль линии),

- `\*\*kwargs` — дополнительные параметры (подписи, легенда).

### \*\*2.2. Примеры\*\*

#### \*\*Простой график\*\*

```python

x = [1, 2, 3, 4]

y = [10, 20, 15, 25]

plt.plot(x, y)

plt.show()

```

\*\*Результат:\*\*

![Линейный график](https://i.imgur.com/XYZ123.png)

#### \*\*Настройки стиля\*\*

```python

plt.plot(x, y, 'r--', linewidth=2, label='Доход') # Красная пунктирная линия

plt.xlabel('Месяц')

plt.ylabel('Сумма')

plt.legend()

plt.show()

```

---

## \*\*3. Метод `.bar()` — столбчатые диаграммы\*\*

Используется для сравнения \*\*категориальных данных\*\*.

### \*\*3.1. Синтаксис\*\*

```python

plt.bar(x, height, width=0.8, \*\*kwargs)

```

- `x` — позиции столбцов,

- `height` — высота столбцов,

- `width` — ширина (по умолчанию 0.8).

### \*\*3.2. Примеры\*\*

#### \*\*Вертикальные столбцы\*\*

```python

categories = ['A', 'B', 'C']

values = [15, 20, 10]

plt.bar(categories, values, color='green')

plt.title('Сравнение категорий')

plt.show()

```

\*\*Результат:\*\*

![Столбчатая диаграмма](https://i.imgur.com/ABC456.png)

#### \*\*Горизонтальные столбцы (`barh`)\*\*

```python

plt.barh(categories, values)

plt.show()

```

---

## \*\*4. Метод `.hist()` — гистограммы\*\*

Используется для визуализации \*\*распределения данных\*\*.

### \*\*4.1. Синтаксис\*\*

```python

plt.hist(data, bins=10, \*\*kwargs)

```

- `data` — входные данные (список/массив),

- `bins` — число интервалов разбиения.

### \*\*4.2. Примеры\*\*

#### \*\*Гистограмма распределения\*\*

```python

data = np.random.normal(0, 1, 1000) # 1000 чисел из N(0,1)

plt.hist(data, bins=30, edgecolor='black')

plt.xlabel('Значение')

plt.ylabel('Частота')

plt.show()

```

\*\*Результат:\*\*

![Гистограмма](https://i.imgur.com/DEF789.png)

#### \*\*Настройка прозрачности\*\*

```python

plt.hist(data, bins=30, alpha=0.5, color='blue')

plt.show()

```

---

## \*\*5. Сравнение методов\*\*

| Метод | Применение | Пример использования |

|----------|--------------------------------|-------------------------------|

| `.plot()`| Тренды, временные ряды | `plt.plot(x, y, 'r-')` |

| `.bar()` | Сравнение категорий | `plt.bar(cats, values)` |

| `.hist()`| Распределение данных | `plt.hist(data, bins=20)` |

---

## \*\*6. Дополнительные возможности\*\*

### \*\*6.1. Несколько графиков на одном поле\*\*

```python

x = [1, 2, 3, 4]

y1 = [10, 20, 15, 25]

y2 = [5, 15, 10, 20]

plt.plot(x, y1, label='Линия 1')

plt.plot(x, y2, label='Линия 2')

plt.legend()

plt.show()

```

### \*\*6.2. Сохранение графика\*\*

```python

plt.plot(x, y)

plt.savefig('graph.png') # Сохранить в файл

```

---

## \*\*Итог\*\*

1. \*\*`.plot()`\*\* — для линейных графиков.

2. \*\*`.bar()`\*\* — для столбчатых диаграмм.

3. \*\*`.hist()`\*\* — для гистограмм распределения.

4. Все методы поддерживают гибкую настройку (цвета, подписи, легенды).

\*\*Финальный пример:\*\*

```python

# Данные

x = np.arange(5)

y = [1, 4, 3, 5, 2]

# Создаем 2 субплта

plt.figure(figsize=(10, 4))

# Линейный график

plt.subplot(1, 2, 1)

plt.plot(x, y, 'go-')

# Столбчатая диаграмма

plt.subplot(1, 2, 2)

plt.bar(x, y)

plt.show()

```

Вопрос №12 Модули. Модуль os.

### \*\*Экзаменационный вопрос: Модули. Модуль `os`\*\*

---

## \*\*1. Понятие модуля в Python\*\*

\*\*Модуль\*\* — это файл с расширением `.py`, содержащий функции, классы и переменные, которые можно импортировать в другие программы.

### \*\*1.1. Зачем нужны модули?\*\*

- \*\*Повторное использование кода\*\* (не нужно писать всё с нуля).

- \*\*Структурирование программ\*\* (разделение на логические части).

- \*\*Стандартная библиотека Python\*\* содержит полезные модули (`os`, `sys`, `math` и др.).

### \*\*1.2. Как импортировать модуль?\*\*

```python

import os # Импорт всего модуля

from os import path # Импорт конкретной функции

import os as operating\_system # Импорт с псевдонимом

```

---

## \*\*2. Модуль `os`\*\*

Модуль `os` предоставляет функции для работы с \*\*операционной системой\*\* (файлы, директории, переменные окружения).

### \*\*2.1. Основные функции модуля `os`\*\*

#### \*\*1. Работа с файлами и директориями\*\*

| Функция/Атрибут | Описание | Пример |

|-----------------------|--------------------------------------------------------------------------|---------------------------------|

| `os.getcwd()` | Возвращает текущую рабочую директорию. | `print(os.getcwd())` |

| `os.chdir(path)` | Изменяет текущую директорию. | `os.chdir("C:/Users")` |

| `os.listdir(path)` | Возвращает список файлов и папок в указанной директории. | `files = os.listdir(".")` |

| `os.mkdir(path)` | Создает новую директорию. | `os.mkdir("new\_folder")` |

| `os.makedirs(path)` | Создает несколько вложенных директорий. | `os.makedirs("dir1/dir2")` |

| `os.remove(path)` | Удаляет файл. | `os.remove("file.txt")` |

| `os.rmdir(path)` | Удаляет пустую директорию. | `os.rmdir("empty\_dir")` |

| `os.rename(src, dst)` | Переименовывает файл/директорию. | `os.rename("old.txt", "new.txt")` |

#### \*\*2. Работа с путями (`os.path`)\*\*

| Функция | Описание | Пример |

|------------------------|--------------------------------------------------------------------------|---------------------------------|

| `os.path.exists(path)` | Проверяет, существует ли файл или директория. | `os.path.exists("file.txt")` |

| `os.path.isfile(path)` | Проверяет, является ли путь файлом. | `os.path.isfile("file.txt")` |

| `os.path.isdir(path)` | Проверяет, является ли путь директорией. | `os.path.isdir("folder")` |

| `os.path.join(...)` | Объединяет части пути с учетом ОС (Windows/Linux). | `os.path.join("dir", "file.txt")` |

| `os.path.abspath(path)`| Возвращает абсолютный путь. | `os.path.abspath("file.txt")` |

#### \*\*3. Переменные окружения\*\*

| Функция | Описание | Пример |

|-----------------------|--------------------------------------------------------------------------|---------------------------------|

| `os.environ` | Словарь с переменными окружения. | `print(os.environ["PATH"])` |

| `os.getenv(key)` | Возвращает значение переменной окружения. | `os.getenv("USERNAME")` |

#### \*\*4. Системные команды\*\*

| Функция | Описание | Пример |

|-----------------------|--------------------------------------------------------------------------|---------------------------------|

| `os.system(command)` | Выполняет команду в терминале ОС. | `os.system("ls -l")` |

| `os.startfile(path)` | Открывает файл в ассоциированной программе (Windows). | `os.startfile("doc.pdf")` |

---

## \*\*3. Примеры использования\*\*

### \*\*3.1. Проверка существования файла\*\*

```python

import os

if os.path.exists("data.txt"):

print("Файл существует!")

else:

print("Файл не найден.")

```

### \*\*3.2. Рекурсивный обход директорий\*\*

```python

for root, dirs, files in os.walk("."):

print(f"Директория: {root}")

print(f"Файлы: {files}")

```

### \*\*3.3. Создание и удаление директории\*\*

```python

os.mkdir("temp") # Создать папку

os.rmdir("temp") # Удалить пустую папку

```

### \*\*3.4. Получение абсолютного пути\*\*

```python

abs\_path = os.path.abspath("script.py")

print(abs\_path) # C:\Users\user\script.py

```

---

## \*\*4. Важные замечания\*\*

- \*\*Кроссплатформенность\*\*:

- `os.path.join()` корректно работает в Windows (`\`) и Linux (`/`).

- Пример:

```python

path = os.path.join("folder", "file.txt") # folder/file.txt (Linux) или folder\file.txt (Windows)

```

- \*\*Безопасность\*\*:

- Проверяйте существование файлов перед удалением (`os.path.exists()`).

- Избегайте `os.system()` для выполнения внешних команд (может быть уязвимостью).

---

## \*\*5. Итог\*\*

1. \*\*Модуль `os`\*\* — инструмент для работы с файлами, директориями и ОС.

2. \*\*Основные функции\*\*:

- Управление файлами: `os.remove()`, `os.rename()`.

- Управление директориями: `os.mkdir()`, `os.listdir()`.

- Работа с путями: `os.path.join()`, `os.path.abspath()`.

- Переменные окружения: `os.environ`, `os.getenv()`.

3. \*\*Всегда проверяйте пути\*\* перед операциями с файлами!

\*\*Финальный пример:\*\*

```python

import os

# Создать папку и файл

os.makedirs("project/data")

with open("project/data.txt", "w") as f:

f.write("Hello, OS module!")

# Проверить размер файла

size = os.path.getsize("project/data.txt")

print(f"Размер файла: {size} байт") # Размер файла: 15 байт

```

Вопрос №13 Работа с файлами. Функция open(). Контекстный менеджер with.

### \*\*Экзаменационный вопрос: Работа с файлами. Функция `open()`. Контекстный менеджер `with`\*\*

---

## \*\*1. Функция `open()`\*\*

Функция `open()` используется для открытия файла в Python. Возвращает файловый объект (file object), через который производится чтение или запись.

### \*\*1.1. Синтаксис\*\*

```python

file = open(file\_path, mode='r', encoding=None)

```

### \*\*1.2. Основные режимы открытия файла\*\*

| Режим | Описание |

|-------|--------------------------------------------------------------------------|

| `'r'` | Чтение (по умолчанию). Файл должен существовать. |

| `'w'` | Запись. Если файл существует, он перезаписывается, иначе создается новый. |

| `'a'` | Дозапись в конец файла. Если файла нет, он создается. |

| `'x'` | Эксклюзивное создание. Если файл существует, вызовет ошибку. |

| `'b'` | Бинарный режим (например, `'rb'` или `'wb'`). |

| `'t'` | Текстовый режим (по умолчанию). |

| `'+'` | Открытие для обновления (чтение + запись). |

### \*\*1.3. Примеры использования\*\*

#### \*\*Чтение файла (`'r'`)\*\*

```python

file = open('example.txt', 'r', encoding='utf-8')

content = file.read() # Чтение всего файла

file.close() # Важно закрыть файл!

print(content)

```

#### \*\*Запись в файл (`'w'`)\*\*

```python

file = open('output.txt', 'w', encoding='utf-8')

file.write("Привет, мир!\n")

file.close() # Сохраняем изменения

```

#### \*\*Дозапись (`'a'`)\*\*

```python

file = open('output.txt', 'a', encoding='utf-8')

file.write("Еще одна строка.\n")

file.close()

```

---

## \*\*2. Контекстный менеджер `with`\*\*

Конструкция `with` автоматически закрывает файл после выполнения блока кода, даже если произошла ошибка.

### \*\*2.1. Синтаксис\*\*

```python

with open(file\_path, mode, encoding) as file:

# Работа с файлом

...

# Файл автоматически закрывается

```

### \*\*2.2. Примеры\*\*

#### \*\*Чтение файла с `with`\*\*

```python

with open('example.txt', 'r', encoding='utf-8') as file:

lines = file.readlines() # Чтение всех строк в список

for line in lines:

print(line.strip()) # Удаляем символы перевода строки

```

#### \*\*Запись в файл с `with`\*\*

```python

with open('output.txt', 'w', encoding='utf-8') as file:

file.write("Первая строка\n")

file.write("Вторая строка\n")

# Файл закрыт автоматически

```

---

## \*\*3. Основные методы файлового объекта\*\*

| Метод | Описание |

|-------------------|--------------------------------------------------------------------------|

| `.read(size)` | Читает `size` символов (если не указано — весь файл). |

| `.readline()` | Читает одну строку. |

| `.readlines()` | Читает все строки и возвращает список. |

| `.write(text)` | Записывает строку `text` в файл. |

| `.writelines(list)` | Записывает список строк в файл (без автоматических `\n`). |

| `.close()` | Закрывает файл (обязательно вызывать после работы!). |

### \*\*Примеры\*\*

#### \*\*Построчное чтение\*\*

```python

with open('data.txt', 'r', encoding='utf-8') as file:

line = file.readline() # Читаем первую строку

while line:

print(line.strip())

line = file.readline() # Читаем следующую строку

```

#### \*\*Запись списка строк\*\*

```python

lines = ["Строка 1\n", "Строка 2\n"]

with open('output.txt', 'w', encoding='utf-8') as file:

file.writelines(lines)

```

---

## \*\*4. Обработка исключений при работе с файлами\*\*

Всегда обрабатывайте возможные ошибки (например, если файл не существует).

```python

try:

with open('missing\_file.txt', 'r', encoding='utf-8') as file:

content = file.read()

except FileNotFoundError:

print("Файл не найден!")

except IOError as e:

print(f"Ошибка ввода-вывода: {e}")

```

---

## \*\*5. Бинарные файлы\*\*

Для работы с изображениями, аудио и другими нетекстовыми данными используйте бинарный режим (`'b'`).

### \*\*Пример чтения/записи бинарного файла\*\*

```python

# Копирование файла в бинарном режиме

with open('image.jpg', 'rb') as src:

with open('copy.jpg', 'wb') as dst:

dst.write(src.read())

```

---

## \*\*6. Итог\*\*

1. \*\*`open()`\*\* — основная функция для работы с файлами.

- Режимы: `'r'` (чтение), `'w'` (запись), `'a'` (дозапись).

- Всегда закрывайте файл (`.close()`) или используйте `with`.

2. \*\*Контекстный менеджер `with`\*\* — безопасный способ работы с файлами.

3. \*\*Методы файлового объекта\*\*:

- `.read()`, `.readline()`, `.readlines()` — для чтения.

- `.write()`, `.writelines()` — для записи.

4. \*\*Обработка исключений\*\* — обязательна для надежности.

\*\*Финальный пример:\*\*

```python

# Чтение и обработка CSV-подобного файла

with open('data.csv', 'r', encoding='utf-8') as file:

for line in file:

values = line.strip().split(',')

print(values)

```

Вопрос №14 Библиотека Pandas. Базовые методы

### \*\*Экзаменационный вопрос: Библиотека Pandas. Базовые методы\*\*

---

## \*\*1. Введение в Pandas\*\*

\*\*Pandas\*\* — это библиотека Python для анализа и обработки данных. Она предоставляет:

- \*\*DataFrame\*\* — табличную структуру данных (аналог Excel).

- \*\*Series\*\* — одномерный массив с метками.

- Удобные методы для чтения, обработки и визуализации данных.

\*\*Импорт библиотеки:\*\*

```python

import pandas as pd

```

---

## \*\*2. Основные структуры данных\*\*

### \*\*2.1. Series\*\*

Одномерный массив с индексами (метками).

```python

s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])

print(s)

```

\*\*Вывод:\*\*

```

a 10

b 20

c 30

dtype: int64

```

### \*\*2.2. DataFrame\*\*

Двумерная таблица (строки и столбцы).

```python

data = {

'Name': ['Alice', 'Bob', 'Charlie'],

'Age': [25, 30, 35],

'City': ['Moscow', 'London', 'Paris']

}

df = pd.DataFrame(data)

print(df)

```

\*\*Вывод:\*\*

```

Name Age City

0 Alice 25 Moscow

1 Bob 30 London

2 Charlie 35 Paris

```

---

## \*\*3. Базовые методы Pandas\*\*

### \*\*3.1. Чтение и запись данных\*\*

| Метод | Описание | Пример |

|---------------------|-----------------------------------|---------------------------------|

| `pd.read\_csv()` | Чтение CSV-файла. | `df = pd.read\_csv('data.csv')` |

| `pd.read\_excel()` | Чтение Excel-файла. | `df = pd.read\_excel('data.xlsx')` |

| `df.to\_csv()` | Сохранение в CSV. | `df.to\_csv('output.csv')` |

| `df.to\_excel()` | Сохранение в Excel. | `df.to\_excel('output.xlsx')` |

### \*\*3.2. Просмотр данных\*\*

| Метод | Описание | Пример |

|---------------------|-----------------------------------|---------------------------------|

| `df.head(n)` | Первые `n` строк (по умолчанию 5).| `df.head(3)` |

| `df.tail(n)` | Последние `n` строк. | `df.tail(2)` |

| `df.info()` | Информация о DataFrame. | `df.info()` |

| `df.describe()` | Статистика по числовым столбцам. | `df.describe()` |

### \*\*3.3. Выбор данных\*\*

| Метод | Описание | Пример |

|---------------------|-----------------------------------|---------------------------------|

| `df['column']` | Выбор столбца. | `df['Name']` |

| `df.loc[row, col]` | Доступ по меткам. | `df.loc[0, 'Age']` |

| `df.iloc[row, col]` | Доступ по индексам. | `df.iloc[1, 2]` |

| `df.query()` | Фильтрация через запрос. | `df.query('Age > 30')` |

### \*\*3.4. Фильтрация данных\*\*

```python

# Выбор строк, где Age > 25

filtered = df[df['Age'] > 25]

print(filtered)

```

\*\*Вывод:\*\*

```

Name Age City

1 Bob 30 London

2 Charlie 35 Paris

```

### \*\*3.5. Добавление и удаление данных\*\*

| Метод | Описание | Пример |

|---------------------|-----------------------------------|---------------------------------|

| `df['new\_col']` | Добавление столбца. | `df['Salary'] = [5000, 6000, 7000]` |

| `df.drop()` | Удаление строк/столбцов. | `df.drop(columns=['City'])` |

### \*\*3.6. Группировка и агрегация\*\*

```python

# Группировка по столбцу и агрегация

grouped = df.groupby('City')['Age'].mean()

print(grouped)

```

\*\*Вывод:\*\*

```

City

London 30

Moscow 25

Paris 35

Name: Age, dtype: int64

```

### \*\*3.7. Обработка пропусков\*\*

| Метод | Описание | Пример |

|---------------------|-----------------------------------|---------------------------------|

| `df.isnull()` | Поиск пропусков. | `df.isnull()` |

| `df.fillna()` | Заполнение пропусков. | `df.fillna(0)` |

| `df.dropna()` | Удаление строк с пропусками. | `df.dropna()` |

---

## \*\*4. Пример комплексного использования\*\*

```python

# Чтение данных

df = pd.read\_csv('students.csv')

# Фильтрация студентов старше 20 лет

adults = df[df['Age'] > 20]

# Группировка по городу и подсчет среднего балла

result = adults.groupby('City')['Score'].mean()

# Сохранение результата

result.to\_csv('result.csv')

```

---

## \*\*5. Итог\*\*

1. \*\*Pandas\*\* предоставляет \*\*DataFrame\*\* и \*\*Series\*\* для работы с данными.

2. \*\*Основные методы\*\*:

- Чтение/запись (`read\_csv`, `to\_csv`).

- Фильтрация (`query`, `loc`).

- Агрегация (`groupby`, `mean`).

- Обработка пропусков (`fillna`, `dropna`).

3. Библиотека оптимизирована для быстрой обработки больших данных.

\*\*Финальный пример:\*\*

```python

# Создание DataFrame

data = {'Product': ['Apple', 'Banana', 'Cherry'], 'Price': [100, 50, 150]}

df = pd.DataFrame(data)

# Добавление столбца

df['Discount'] = df['Price'] \* 0.1

# Вывод результата

print(df)

```

\*\*Вывод:\*\*

```

Product Price Discount

0 Apple 100 10.0

1 Banana 50 5.0

2 Cherry 150 15.0

```

Вопрос №15 Библиотека Pandas. Статистические методы.

### \*\*Экзаменационный вопрос: Библиотека Pandas. Статистические методы\*\*

---

## \*\*1. Основные статистические методы в Pandas\*\*

Библиотека Pandas предоставляет широкий набор методов для статистического анализа данных. Эти методы применяются к \*\*DataFrame\*\* и \*\*Series\*\* и позволяют быстро получать описательные статистики, вычислять корреляции, агрегировать данные и многое другое.

---

## \*\*2. Описательные статистики\*\*

### \*\*2.1. Базовые методы\*\*

| Метод | Описание | Пример |

|---------------------|--------------------------------------------------------------------------|---------------------------------|

| `.describe()` | Сводная статистика (среднее, мин/макс, квартили и др.). | `df.describe()` |

| `.mean()` | Среднее значение. | `df['Age'].mean()` |

| `.median()` | Медиана. | `df['Salary'].median()` |

| `.mode()` | Мода (наиболее часто встречающееся значение). | `df['City'].mode()` |

| `.min()`, `.max()` | Минимальное и максимальное значения. | `df['Price'].min()` |

| `.std()` | Стандартное отклонение. | `df['Height'].std()` |

| `.var()` | Дисперсия. | `df['Weight'].var()` |

| `.quantile(q)` | Квантиль (по умолчанию `q=0.5` — медиана). | `df['Income'].quantile(0.75)` |

| `.sum()` | Сумма значений. | `df['Sales'].sum()` |

| `.count()` | Количество непустых значений. | `df['Name'].count()` |

#### \*\*Пример использования `.describe()`\*\*

```python

import pandas as pd

data = {'Age': [25, 30, 35, 40, 45], 'Salary': [50000, 60000, 70000, 80000, 90000]}

df = pd.DataFrame(data)

print(df.describe())

```

\*\*Вывод:\*\*

```

Age Salary

count 5.000000 5.000000

mean 35.000000 70000.000000

std 7.905694 15811.388301

min 25.000000 50000.000000

25% 30.000000 60000.000000

50% 35.000000 70000.000000

75% 40.000000 80000.000000

max 45.000000 90000.000000

```

---

## \*\*3. Корреляции и ковариации\*\*

### \*\*3.1. Корреляция (`.corr()`)\*\*

Метод `.corr()` вычисляет \*\*коэффициент корреляции Пирсона\*\* между столбцами (значения от -1 до 1):

- \*\*1\*\* — сильная прямая зависимость,

- \*\*-1\*\* — сильная обратная зависимость,

- \*\*0\*\* — отсутствие линейной зависимости.

```python

print(df.corr())

```

\*\*Вывод:\*\*

```

Age Salary

Age 1.000000 1.000000

Salary 1.000000 1.000000

```

\*(В данном примере корреляция = 1, так как данные линейно зависимы.)\*

### \*\*3.2. Ковариация (`.cov()`)\*\*

Показывает, как изменяются две переменные вместе:

```python

print(df.cov())

```

\*\*Вывод:\*\*

```

Age Salary

Age 62.5 125000.0

Salary 125000.0 250000000.0

```

---

## \*\*4. Агрегация данных (`.groupby()`, `.agg()`)\*\*

### \*\*4.1. Группировка и агрегация\*\*

Метод `.groupby()` группирует данные по заданному столбцу, а `.agg()` применяет несколько статистических функций.

```python

data = {

'City': ['Moscow', 'London', 'Moscow', 'London', 'Moscow'],

'Sales': [100, 200, 150, 300, 250]

}

df = pd.DataFrame(data)

# Группировка по городу и агрегация

result = df.groupby('City')['Sales'].agg(['sum', 'mean', 'count'])

print(result)

```

\*\*Вывод:\*\*

```

sum mean count

City

London 500 250.0 2

Moscow 500 166.666667 3

```

### \*\*4.2. Пользовательские агрегации\*\*

Можно передавать несколько функций или лямбда-выражения:

```python

result = df.groupby('City')['Sales'].agg([

('Total', 'sum'),

('Average', lambda x: x.mean()),

('Range', lambda x: x.max() - x.min())

])

```

---

## \*\*5. Работа с пропусками (`.isnull()`, `.fillna()`)\*\*

### \*\*5.1. Поиск пропусков\*\*

```python

df.isnull().sum() # Количество пропусков в каждом столбце

```

### \*\*5.2. Заполнение пропусков\*\*

```python

# Заполнение средним значением

df['Age'].fillna(df['Age'].mean(), inplace=True)

# Заполнение модой

df['City'].fillna(df['City'].mode()[0], inplace=True)

```

---

## \*\*6. Другие полезные методы\*\*

| Метод | Описание | Пример |

|---------------------|--------------------------------------------------------------------------|---------------------------------|

| `.value\_counts()` | Подсчет уникальных значений. | `df['City'].value\_counts()` |

| `.unique()` | Уникальные значения. | `df['Product'].unique()` |

| `.nunique()` | Количество уникальных значений. | `df['Customer'].nunique()` |

| `.pct\_change()` | Процентное изменение между элементами. | `df['Price'].pct\_change()` |

| `.cumsum()`, `.cumprod()` | Кумулятивная сумма/произведение. | `df['Sales'].cumsum()` |

---

## \*\*7. Пример комплексного анализа\*\*

```python

# Чтение данных

df = pd.read\_csv('sales\_data.csv')

# Описательная статистика

print(df.describe())

# Корреляция между продажами и прибылью

print(df[['Sales', 'Profit']].corr())

# Группировка по региону

region\_stats = df.groupby('Region')['Sales'].agg(['sum', 'mean', 'std'])

# Заполнение пропусков в возрасте

df['Age'].fillna(df['Age'].median(), inplace=True)

```

---

## \*\*8. Итог\*\*

1. \*\*Описательные статистики\*\*:

- `.describe()`, `.mean()`, `.median()`, `.std()`, `.quantile()`.

2. \*\*Корреляция и ковариация\*\*:

- `.corr()`, `.cov()`.

3. \*\*Агрегация данных\*\*:

- `.groupby()`, `.agg()`.

4. \*\*Работа с пропусками\*\*:

- `.isnull()`, `.fillna()`.

5. \*\*Другие методы\*\*:

- `.value\_counts()`, `.unique()`, `.cumsum()`.

\*\*Финальный пример:\*\*

```python

# Анализ данных о студентах

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

'Score': [85, 90, 78, 92, 88],

'Age': [20, 21, 22, 20, 21]

}

df = pd.DataFrame(data)

# Статистика по возрасту и оценкам

print(df[['Age', 'Score']].describe())

# Корреляция между возрастом и оценками

print(df[['Age', 'Score']].corr())

```

\*\*Вывод:\*\*

```

Age Score

count 5.000000 5.000000

mean 20.800000 86.600000

std 0.836660 5.683309

min 20.000000 78.000000

25% 20.000000 85.000000

50% 21.000000 88.000000

75% 21.000000 90.000000

max 22.000000 92.000000

Age Score

Age 1.000000 -0.329573

Score -0.329573 1.000000

```

Вопрос № 16 REST API. Основные методы: GET, POST

### \*\*Экзаменационный вопрос: REST API. Основные методы: GET, POST\*\*

---

## \*\*1. Что такое REST API?\*\*

\*\*REST API\*\* (Representational State Transfer Application Programming Interface) — это архитектурный стиль взаимодействия между клиентом и сервером через HTTP-запросы.

- Использует стандартные HTTP-методы (`GET`, `POST`, `PUT`, `DELETE` и др.).

- Данные передаются в форматах \*\*JSON\*\* или \*\*XML\*\*.

- Не сохраняет состояние (\*\*stateless\*\*): каждый запрос содержит всю необходимую информацию.

---

## \*\*2. Основные HTTP-методы\*\*

### \*\*2.1. GET\*\*

- \*\*Назначение\*\*: Получение данных с сервера.

- \*\*Идемпотентность\*\*: Да (повторные запросы не изменяют состояние сервера).

- \*\*Безопасность\*\*: Да (не изменяет данные на сервере).

- \*\*Пример\*\*:

```http

GET /api/users/1 HTTP/1.1

Host: example.com

```

\*\*Ответ (JSON):\*\*

```json

{

"id": 1,

"name": "Alice",

"email": "alice@example.com"

}

```

### \*\*2.2. POST\*\*

- \*\*Назначение\*\*: Создание новых данных на сервере.

- \*\*Идемпотентность\*\*: Нет (повторные запросы могут создавать дубликаты).

- \*\*Безопасность\*\*: Нет (изменяет данные на сервере).

- \*\*Пример\*\*:

```http

POST /api/users HTTP/1.1

Host: example.com

Content-Type: application/json

{

"name": "Bob",

"email": "bob@example.com"

}

```

\*\*Ответ (JSON):\*\*

```json

{

"id": 2,

"name": "Bob",

"email": "bob@example.com"

}

```

---

## \*\*3. Ключевые различия GET и POST\*\*

| Критерий | GET | POST |

|-------------------|------------------------------|------------------------------|

| \*\*Назначение\*\* | Получение данных | Создание/отправка данных |

| \*\*Данные в запросе\*\* | Передаются в URL (`?key=value`) | Передаются в теле запроса (body) |

| \*\*Кэширование\*\* | Да | Нет |

| \*\*Ограничение длины\*\* | Да (макс. длина URL) | Нет |

| \*\*Использование\*\* | Поиск, фильтрация | Регистрация, оплата, формы |

---

## \*\*4. Примеры кода (Python)\*\*

### \*\*4.1. GET-запрос с `requests`\*\*

```python

import requests

response = requests.get('https://api.example.com/users/1')

if response.status\_code == 200:

user = response.json()

print(user['name']) # Alice

```

### \*\*4.2. POST-запрос с `requests`\*\*

```python

data = {'name': 'Bob', 'email': 'bob@example.com'}

response = requests.post('https://api.example.com/users', json=data)

if response.status\_code == 201:

new\_user = response.json()

print(new\_user['id']) # 2

```

---

## \*\*5. Статусные коды HTTP\*\*

- \*\*200 OK\*\*: Успешный GET-запрос.

- \*\*201 Created\*\*: Успешный POST-запрос (ресурс создан).

- \*\*400 Bad Request\*\*: Неверный синтаксис запроса.

- \*\*404 Not Found\*\*: Ресурс не найден.

- \*\*500 Internal Server Error\*\*: Ошибка сервера.

---

## \*\*6. Итог\*\*

1. \*\*GET\*\* — для получения данных (без изменений на сервере).

2. \*\*POST\*\* — для создания новых данных (передача в теле запроса).

3. REST API использует JSON/XML для обмена данными.

4. Важно проверять \*\*HTTP-статусы\*\* и \*\*обрабатывать ошибки\*\*.

\*\*Финальный пример:\*\*

```python

# Получение списка пользователей (GET)

users = requests.get('https://api.example.com/users').json()

# Добавление нового пользователя (POST)

new\_user = {'name': 'Charlie', 'email': 'charlie@example.com'}

requests.post('https://api.example.com/users', json=new\_user)

```

Вопрос № 17 Fast API. Создание и запуск приложения с помощью данного фреймворка

### \*\*Экзаменационный вопрос: FastAPI. Создание и запуск приложения\*\*

---

## \*\*1. Что такое FastAPI?\*\*

\*\*FastAPI\*\* — современный фреймворк для создания \*\*RESTful API\*\* на Python, отличающийся:

- Высокой производительностью (на основе \*\*Starlette\*\* и \*\*Pydantic\*\*).

- Автоматической генерацией \*\*OpenAPI\*\*-документации.

- Поддержкой \*\*асинхронности\*\* (async/await).

---

## \*\*2. Установка FastAPI\*\*

Перед началом работы установите FastAPI и сервер \*\*Uvicorn\*\* (ASGI-сервер для запуска):

```bash

pip install fastapi uvicorn

```

---

## \*\*3. Создание базового приложения\*\*

### \*\*3.1. Минимальное приложение\*\*

Создайте файл `main.py`:

```python

from fastapi import FastAPI

app = FastAPI() # Создание экземпляра приложения

@app.get("/") # Декоратор для GET-запроса

def read\_root():

return {"message": "Hello, World!"}

```

### \*\*3.2. Запуск приложения\*\*

Запустите сервер через Uvicorn:

```bash

uvicorn main:app --reload

```

- `main:app` — файл `main.py` и переменная `app` в нём.

- `--reload` — автоматическая перезагрузка при изменении кода (только для разработки).

После запуска откройте в браузере:

- http://127.0.0.1:8000 — увидите `{"message":"Hello, World!"}`.

- http://127.0.0.1:8000/docs — автоматическая документация Swagger.

---

## \*\*4. Добавление новых эндпоинтов\*\*

### \*\*4.1. GET-запрос с параметрами\*\*

```python

@app.get("/items/{item\_id}")

def read\_item(item\_id: int, q: str = None):

return {"item\_id": item\_id, "q": q}

```

- \*\*Пример запроса\*\*:

http://127.0.0.1:8000/items/42?q=test

### \*\*4.2. POST-запрос с телом\*\*

```python

from pydantic import BaseModel

class Item(BaseModel):

name: str

price: float

@app.post("/items/")

def create\_item(item: Item):

return {"item\_name": item.name, "item\_price": item.price}

```

- \*\*Пример тела запроса (JSON)\*\*:

```json

{"name": "Laptop", "price": 999.99}

```

---

## \*\*5. Автоматическая документация\*\*

FastAPI генерирует две версии документации:

1. \*\*Swagger UI\*\*: http://127.0.0.1:8000/docs

2. \*\*ReDoc\*\*: http://127.0.0.1:8000/redoc

Документация включает:

- Описание всех эндпоинтов.

- Возможность тестирования API прямо в браузере.

- Схемы данных (Pydantic-модели).

---

## \*\*6. Пример с асинхронностью\*\*

FastAPI поддерживает асинхронные функции:

```python

@app.get("/async-example")

async def async\_endpoint():

await some\_async\_operation()

return {"status": "done"}

```

---

## \*\*7. Запуск в продакшене\*\*

Для продакшн-сервера используйте:

```bash

uvicorn main:app --host 0.0.0.0 --port 80

```

- `--host 0.0.0.0` — доступ с любого IP.

- `--port 80` — стандартный HTTP-порт.

Для масштабирования добавьте \*\*Gunicorn\*\* (для Unix-систем):

```bash

gunicorn -w 4 -k uvicorn.workers.UvicornWorker main:app

```

---

## \*\*8. Итог\*\*

1. \*\*Установка\*\*:

```bash

pip install fastapi uvicorn

```

2. \*\*Создание приложения\*\*:

- Объявите `app = FastAPI()`.

- Добавьте эндпоинты через декораторы (`@app.get`, `@app.post`).

3. \*\*Запуск\*\*:

```bash

uvicorn main:app --reload

```

4. \*\*Документация\*\*: Доступна по адресу `/docs` и `/redoc`.

5. \*\*Продакшен\*\*: Используйте Uvicorn + Gunicorn.

\*\*Финальный пример (main.py):\*\*

```python

from fastapi import FastAPI

from pydantic import BaseModel

app = FastAPI()

class User(BaseModel):

username: str

email: str

@app.get("/")

def home():

return {"status": "OK"}

@app.post("/users/")

def create\_user(user: User):

return {"username": user.username, "email": user.email}

``

Вопрос № 18 Активационные функции.

### \*\*Экзаменационный вопрос: Активационные функции\*\*

---

## \*\*1. Что такое активационная функция?\*\*

\*\*Активационная функция\*\* — это математическая функция, применяемая к выходному сигналу нейрона в искусственной нейронной сети. Она определяет, будет ли нейрон активирован (передаст сигнал дальше) и насколько сильным будет этот сигнал.

\*\*Основные задачи:\*\*

- Введение \*\*нелинейности\*\* в сеть (без этого сеть становится линейным классификатором).

- Ограничение выхода нейрона в определённом диапазоне (например, [0, 1] или [-1, 1]).

---

## \*\*2. Виды активационных функций\*\*

### \*\*2.1. Линейная (Linear)\*\*

- \*\*Формула\*\*:

\[

f(x) = x

\]

- \*\*Применение\*\*:

Используется редко, так как не добавляет нелинейности.

- \*\*График\*\*:

![Linear](https://i.imgur.com/XYZ123.png)

### \*\*2.2. Сигмоида (Sigmoid)\*\*

- \*\*Формула\*\*:

\[

f(x) = \frac{1}{1 + e^{-x}}

\]

- \*\*Диапазон\*\*: (0, 1).

- \*\*Применение\*\*:

- Выходной слой бинарных классификаторов (вероятность).

- Исторически использовалась в скрытых слоях (сейчас реже).

- \*\*Проблемы\*\*:

- \*\*Проблема исчезающего градиента\*\* (градиент близок к 0 при больших |x|).

- Не симметрична относительно 0.

- \*\*График\*\*:

![Sigmoid](https://i.imgur.com/ABC456.png)

### \*\*2.3. Гиперболический тангенс (Tanh)\*\*

- \*\*Формула\*\*:

\[

f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}

\]

- \*\*Диапазон\*\*: (-1, 1).

- \*\*Применение\*\*:

- Скрытые слои (лучше сигмоиды из-за симметричности).

- \*\*Проблемы\*\*:

- Также страдает от проблемы исчезающего градиента.

- \*\*График\*\*:

![Tanh](https://i.imgur.com/DEF789.png)

### \*\*2.4. ReLU (Rectified Linear Unit)\*\*

- \*\*Формула\*\*:

\[

f(x) = \max(0, x)

\]

- \*\*Диапазон\*\*: [0, +∞).

- \*\*Применение\*\*:

- Основная функция для скрытых слоёв в современных сетях.

- \*\*Преимущества\*\*:

- Устраняет проблему исчезающего градиента для положительных \( x \).

- Быстро вычисляется.

- \*\*Проблемы\*\*:

- \*\*"Умирающий ReLU"\*\* (нейроны могут "отключаться" при \( x < 0 \)).

- \*\*График\*\*:

![ReLU](https://i.imgur.com/GHI012.png)

### \*\*2.5. Leaky ReLU\*\*

- \*\*Формула\*\*:

\[

f(x) = \begin{cases}

x, & \text{если } x > 0 \\

\alpha x, & \text{иначе}

\end{cases}

\]

где \( \alpha \) — малый коэффициент (например, 0.01).

- \*\*Применение\*\*:

- Альтернатива ReLU для избежания "умирающих" нейронов.

- \*\*График\*\*:

![Leaky ReLU](https://i.imgur.com/JKL345.png)

### \*\*2.6. Softmax\*\*

- \*\*Формула\*\*:

\[

f(x\_i) = \frac{e^{x\_i}}{\sum\_{j=1}^n e^{x\_j}}

\]

- \*\*Диапазон\*\*: (0, 1), сумма выходов = 1.

- \*\*Применение\*\*:

- Выходной слой \*\*многоклассового классификатора\*\* (вероятности классов).

---

## \*\*3. Сравнение функций\*\*

| Функция | Диапазон | Нелинейность | Плюсы | Минусы |

|---------------|-------------|--------------|--------------------------------|----------------------------|

| \*\*Linear\*\* | (-∞, +∞) | Нет | Простота | Бесполезна для глубоких сетей |

| \*\*Sigmoid\*\* | (0, 1) | Да | Интерпретируемость (вероятность)| Исчезающий градиент |

| \*\*Tanh\*\* | (-1, 1) | Да | Симметричность | Исчезающий градиент |

| \*\*ReLU\*\* | [0, +∞) | Да | Быстрая, устраняет градиент | "Умирающий ReLU" |

| \*\*Leaky ReLU\*\*| (-∞, +∞) | Да | Решает проблему "умирания" | Нужен параметр \( \alpha \) |

| \*\*Softmax\*\* | (0, 1) | Да | Нормализация вероятностей | Только для выходного слоя |

---

## \*\*4. Какую функцию выбрать?\*\*

- \*\*Скрытые слои\*\*: ReLU / Leaky ReLU (высокая скорость обучения).

- \*\*Выходной слой\*\*:

- \*\*Бинарная классификация\*\*: Сигмоида.

- \*\*Многоклассовая классификация\*\*: Softmax.

- \*\*Регрессия\*\*: Линейная.

---

## \*\*5. Примеры в коде (PyTorch)\*\*

```python

import torch

import torch.nn as nn

# Пример слоёв с разными функциями

model = nn.Sequential(

nn.Linear(10, 20),

nn.ReLU(), # Скрытый слой

nn.Linear(20, 5),

nn.Softmax(dim=1) # Выходной слой

)

input = torch.randn(1, 10)

output = model(input)

print(output)

```

---

## \*\*6. Итог\*\*

1. Активационные функции добавляют \*\*нелинейность\*\* в нейросети.

2. \*\*ReLU\*\* — стандарт для скрытых слоёв.

3. \*\*Softmax\*\* и \*\*сигмоида\*\* используются в выходных слоях классификаторов.

4. Проблемы:

- Исчезающий градиент (сигмоида, tanh).

- "Умирающий ReLU" (решается через Leaky ReLU).

\*\*Финальный выбор:\*\*

- Для глубоких сетей: \*\*ReLU\*\* / \*\*Leaky ReLU\*\*.

* Для вероятностей: \*\*Softmax\*\* (многокласс) или \*\*Sigmoid\*\* (бинар).

Вопрос № 19 Функция ошибки. Стандартные функции ошибок keras.

### \*\*Экзаменационный вопрос: Функция ошибки. Стандартные функции ошибок в Keras\*\*

---

## \*\*1. Что такое функция ошибки (loss function)?\*\*

\*\*Функция ошибки\*\* (или \*\*функция потерь\*\*) — это мера, которая оценивает, насколько предсказания модели отличаются от истинных значений.

- \*\*Цель обучения\*\*: Минимизировать значение функции ошибки.

- \*\*Использование\*\*:

- В \*\*Keras\*\* функция ошибки задаётся при компиляции модели:

```python

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

```

---

## \*\*2. Стандартные функции ошибок в Keras\*\*

### \*\*2.1. Для задач регрессии\*\*

#### \*\*1. Mean Squared Error (MSE)\*\*

- \*\*Формула\*\*:

\[

\text{MSE} = \frac{1}{n} \sum\_{i=1}^n (y\_i - \hat{y}\_i)^2

\]

- \*\*Применение\*\*: Предсказание непрерывных значений (например, цены, температуры).

- \*\*Keras\*\*:

```python

model.compile(loss='mean\_squared\_error', optimizer='sgd')

```

#### \*\*2. Mean Absolute Error (MAE)\*\*

- \*\*Формула\*\*:

\[

\text{MAE} = \frac{1}{n} \sum\_{i=1}^n |y\_i - \hat{y}\_i|

\]

- \*\*Применение\*\*: Менее чувствительна к выбросам, чем MSE.

- \*\*Keras\*\*:

```python

model.compile(loss='mean\_absolute\_error', optimizer='sgd')

```

---

### \*\*2.2. Для задач классификации\*\*

#### \*\*1. Binary Cross-Entropy\*\*

- \*\*Формула\*\*:

\[

\text{BCE} = -\frac{1}{n} \sum\_{i=1}^n \left[ y\_i \log(\hat{y}\_i) + (1 - y\_i) \log(1 - \hat{y}\_i) \right]

\]

- \*\*Применение\*\*: Бинарная классификация (например, спам/не спам).

- \*\*Keras\*\*:

```python

model.compile(loss='binary\_crossentropy', optimizer='adam')

```

#### \*\*2. Categorical Cross-Entropy\*\*

- \*\*Формула\*\*:

\[

\text{CCE} = -\frac{1}{n} \sum\_{i=1}^n \sum\_{j=1}^m y\_{ij} \log(\hat{y}\_{ij})

\]

- \*\*Применение\*\*: Многоклассовая классификация (например, распознавание цифр MNIST).

- \*\*Требование\*\*: Метки должны быть в \*\*one-hot encoding\*\*.

- \*\*Keras\*\*:

```python

model.compile(loss='categorical\_crossentropy', optimizer='adam')

```

#### \*\*3. Sparse Categorical Cross-Entropy\*\*

- \*\*Отличие от CCE\*\*: Метки — целые числа (не one-hot).

- \*\*Keras\*\*:

```python

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam')

```

---

### \*\*2.3. Для других задач\*\*

#### \*\*1. Huber Loss\*\*

- \*\*Формула\*\*:

\[

L\_\delta = \begin{cases}

\frac{1}{2} (y - \hat{y})^2, & \text{если } |y - \hat{y}| \leq \delta \\

\delta \cdot (|y - \hat{y}| - \frac{1}{2} \delta), & \text{иначе}

\end{cases}

\]

- \*\*Применение\*\*: Регрессия, устойчивая к выбросам.

- \*\*Keras\*\*:

```python

model.compile(loss='huber\_loss', optimizer='adam')

```

#### \*\*2. KL Divergence\*\*

- \*\*Применение\*\*: Для задач, где нужно сравнить два распределения (например, в VAEs).

- \*\*Keras\*\*:

```python

model.compile(loss='kl\_divergence', optimizer='adam')

```

---

## \*\*3. Как выбрать функцию ошибки?\*\*

| Задача | Функция ошибки | Пример использования |

|----------------------------|----------------------------------------|----------------------------------|

| \*\*Регрессия\*\* | MSE, MAE, Huber | Предсказание цен |

| \*\*Бинарная классификация\*\* | Binary Cross-Entropy | Спам/не спам |

| \*\*Многоклассовая классификация\*\* | Categorical Cross-Entropy (one-hot) или Sparse Categorical Cross-Entropy (целые метки) | Распознавание цифр (MNIST) |

| \*\*Устойчивая к выбросам регрессия\*\* | Huber Loss | Прогнозирование температуры |

---

## \*\*4. Примеры в Keras\*\*

### \*\*4.1. Регрессия (MSE)\*\*

```python

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model = Sequential([

Dense(64, activation='relu', input\_shape=(10,)),

Dense(1) # Выходной слой для регрессии

])

model.compile(loss='mean\_squared\_error', optimizer='adam')

```

### \*\*4.2. Бинарная классификация (Binary Cross-Entropy)\*\*

```python

model = Sequential([

Dense(64, activation='relu', input\_shape=(20,)),

Dense(1, activation='sigmoid') # Бинарный выход

])

model.compile(loss='binary\_crossentropy', optimizer='adam')

```

### \*\*4.3. Многоклассовая классификация (Categorical Cross-Entropy)\*\*

```python

model = Sequential([

Dense(64, activation='relu', input\_shape=(784,)),

Dense(10, activation='softmax') # 10 классов

])

model.compile(loss='categorical\_crossentropy', optimizer='adam')

```

---

## \*\*5. Важные замечания\*\*

1. \*\*Активационные функции\*\* должны соответствовать задаче:

- Регрессия: линейная активация (`activation='linear'`).

- Бинарная классификация: сигмоида (`activation='sigmoid'`).

- Многоклассовая классификация: softmax (`activation='softmax'`).

2. Для \*\*imbalanced данных\*\* (несбалансированных классов) можно использовать:

- Взвешенные версии cross-entropy.

- Focal Loss (для борьбы с "лёгкими" примерами).

---

## \*\*6. Итог\*\*

1. \*\*Функция ошибки\*\* — ключевой компонент обучения модели.

2. \*\*Основные функции в Keras\*\*:

- Регрессия: `MSE`, `MAE`, `Huber`.

- Классификация: `Binary Cross-Entropy`, `Categorical Cross-Entropy`.

3. Выбор зависит от \*\*типа задачи\*\* и \*\*формата данных\*\*.

\*\*Финальный пример (многоклассовая классификация):\*\*

```python

model = Sequential([

Dense(128, activation='relu', input\_shape=(784,)),

Dense(10, activation='softmax')

])

model.compile(

loss='sparse\_categorical\_crossentropy', # Для целых меток

optimizer='adam',

metrics=['accuracy']

)

```

Вопрос № 20 Оптимизаторы. Стандартные оптимизаторы keras.

### \*\*Экзаменационный вопрос: Оптимизаторы. Стандартные оптимизаторы Keras\*\*

---

## \*\*1. Что такое оптимизатор?\*\*

\*\*Оптимизатор\*\* — это алгоритм, который корректирует веса нейронной сети для минимизации функции потерь (loss function) в процессе обучения.

- \*\*Задача\*\*: Найти глобальный минимум функции ошибки.

- \*\*Как работает\*\*: Вычисляет градиенты потерь относительно весов и обновляет веса с учётом learning rate.

В \*\*Keras\*\* оптимизатор задаётся при компиляции модели:

```python

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

```

---

## \*\*2. Стандартные оптимизаторы в Keras\*\*

### \*\*2.1. SGD (Stochastic Gradient Descent)\*\*

- \*\*Формула обновления весов\*\*:

\[

w\_{t+1} = w\_t - \eta \cdot \nabla L(w\_t)

\]

где:

- \( \eta \) — learning rate,

- \( \nabla L(w\_t) \) — градиент функции потерь.

- \*\*Параметры\*\*:

```python

tf.keras.optimizers.SGD(

learning\_rate=0.01,

momentum=0.0, # Импульс (ускоряет сходимость)

nesterov=False # Улучшенный импульс (Nesterov)

)

```

- \*\*Плюсы\*\*:

- Простота.

- Легко интерпретировать.

- \*\*Минусы\*\*:

- Медленная сходимость на сложных ландшафтах.

- Чувствительность к learning rate.

- \*\*Когда использовать\*\*:

- Для простых задач или как базовый вариант.

---

### \*\*2.2. RMSprop\*\*

- \*\*Идея\*\*: Адаптивный learning rate для каждого параметра (уменьшает шаг для больших градиентов).

- \*\*Формула\*\*:

\[

w\_{t+1} = w\_t - \frac{\eta}{\sqrt{E[g^2]\_t + \epsilon}} \cdot g\_t

\]

где \( E[g^2]\_t \) — скользящее среднее квадратов градиентов.

- \*\*Параметры\*\*:

```python

tf.keras.optimizers.RMSprop(

learning\_rate=0.001,

rho=0.9, # Коэффициент затухания

epsilon=1e-07

)

```

- \*\*Плюсы\*\*:

- Хорош для RNN и задач с noisy градиентами.

- \*\*Минусы\*\*:

- Может "застревать" в локальных минимумах.

- \*\*Когда использовать\*\*:

- Рекуррентные сети (RNN).

---

### \*\*2.3. Adam (Adaptive Moment Estimation)\*\*

- \*\*Идея\*\*: Комбинация \*\*Momentum\*\* и \*\*RMSprop\*\*.

- \*\*Формула\*\*:

\[

w\_{t+1} = w\_t - \frac{\eta \cdot \hat{m}\_t}{\sqrt{\hat{v}\_t} + \epsilon}

\]

где:

- \( \hat{m}\_t \) — оценка первого момента (среднее градиентов),

- \( \hat{v}\_t \) — оценка второго момента (дисперсия градиентов).

- \*\*Параметры\*\*:

```python

tf.keras.optimizers.Adam(

learning\_rate=0.001,

beta\_1=0.9, # Для первого момента (среднее)

beta\_2=0.999, # Для второго момента (дисперсия)

epsilon=1e-07

)

```

- \*\*Плюсы\*\*:

- Автоматическая настройка learning rate.

- Быстрая сходимость.

- \*\*Минусы\*\*:

- Может переобучаться на маленьких датасетах.

- \*\*Когда использовать\*\*:

- \*\*По умолчанию\*\* для большинства задач (CNN, DNN).

---

### \*\*2.4. Adagrad (Adaptive Gradient Algorithm)\*\*

- \*\*Идея\*\*: Уменьшает learning rate для часто обновляемых параметров.

- \*\*Формула\*\*:

\[

w\_{t+1} = w\_t - \frac{\eta}{\sqrt{G\_t + \epsilon}} \cdot g\_t

\]

где \( G\_t \) — сумма квадратов прошлых градиентов.

- \*\*Параметры\*\*:

```python

tf.keras.optimizers.Adagrad(

learning\_rate=0.01,

initial\_accumulator\_value=0.1,

epsilon=1e-07

)

```

- \*\*Плюсы\*\*:

- Хорош для sparse данных (например, NLP).

- \*\*Минусы\*\*:

- Learning rate может стать слишком маленьким.

- \*\*Когда использовать\*\*:

- Работа с текстами (Word2Vec, GloVe).

---

### \*\*2.5. Nadam (Nesterov-accelerated Adam)\*\*

- \*\*Идея\*\*: Adam + Nesterov momentum.

- \*\*Плюсы\*\*:

- Ещё быстрее, чем Adam.

- \*\*Когда использовать\*\*:

- Для ускорения обучения глубоких сетей.

---

## \*\*3. Сравнение оптимизаторов\*\*

| Оптимизатор | Адаптивный LR? | Momentum? | Когда использовать |

|--------------|----------------|-----------|----------------------------------------|

| \*\*SGD\*\* | ❌ | ❌ (опционально) | Базовые задачи |

| \*\*RMSprop\*\* | ✅ | ❌ | RNN, зашумлённые данные |

| \*\*Adam\*\* | ✅ | ✅ | \*\*Универсальный выбор\*\* (CNN, DNN) |

| \*\*Adagrad\*\* | ✅ | ❌ | Sparse данные (NLP) |

| \*\*Nadam\*\* | ✅ | ✅ (Nesterov) | Глубокие сети |

---

## \*\*4. Как выбрать оптимизатор?\*\*

1. \*\*Начните с Adam\*\* — работает в 90% случаев.

2. \*\*Для RNN\*\* попробуйте RMSprop.

3. \*\*Для NLP\*\* — Adagrad.

4. \*\*Если Adam переобучается\*\* — используйте SGD + momentum.

---

## \*\*5. Примеры в Keras\*\*

### \*\*5.1. Использование встроенного оптимизатора\*\*

```python

model.compile(

optimizer='adam', # Встроенный Adam

loss='categorical\_crossentropy'

)

```

### \*\*5.2. Настройка параметров оптимизатора\*\*

```python

from tensorflow.keras.optimizers import Adam

custom\_adam = Adam(

learning\_rate=0.0001,

beta\_1=0.9,

beta\_2=0.999

)

model.compile(

optimizer=custom\_adam,

loss='mse'

)

```

---

## \*\*6. Итог\*\*

1. \*\*Adam\*\* — лучший выбор по умолчанию.

2. \*\*SGD\*\* — для контроля за learning rate.

3. \*\*RMSprop\*\* — для RNN.

4. \*\*Adagrad\*\* — для NLP.

\*\*Финальный пример:\*\*

```python

model = Sequential([

Dense(64, activation='relu', input\_shape=(784,)),

Dense(10, activation='softmax')

])

model.compile(

optimizer=Adam(learning\_rate=0.001),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy']

)

```

Вопрос № 21 Обучение нейронной сети. Выборки. Валидация данных.

### \*\*Экзаменационный вопрос: Обучение нейронной сети. Выборки. Валидация данных\*\*

---

## \*\*1. Этапы работы с данными\*\*

Перед обучением нейронной сети данные разделяются на \*\*три выборки\*\*:

1. \*\*Обучающая выборка (Training set)\*\* — используется для обучения модели.

2. \*\*Валидационная выборка (Validation set)\*\* — для подбора гиперпараметров и контроля переобучения.

3. \*\*Тестовая выборка (Test set)\*\* — для финальной оценки качества модели.

\*\*Стандартное разделение\*\*:

- 70% — обучение,

- 15% — валидация,

- 15% — тестирование.

\*\*Пример в коде (Sklearn):\*\*

```python

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

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

```

---

## \*\*2. Валидация данных\*\*

\*\*Валидация\*\* — процесс контроля качества модели на данных, которые не участвовали в обучении.

### \*\*2.1. Методы валидации\*\*

#### \*\*1. Hold-out Validation\*\*

- Данные делятся на \*\*train/val/test\*\* один раз.

- \*\*Плюсы\*\*: Простота.

- \*\*Минусы\*\*: Зависит от случайного разбиения.

#### \*\*2. K-Fold Cross-Validation\*\*

- Данные разбиваются на \*\*K частей\*\* (например, 5).

- Модель обучается \*\*K раз\*\*, каждый раз на разных блоках.

- \*\*Плюсы\*\*: Стабильная оценка качества.

- \*\*Минусы\*\*: Вычислительно затратно.

\*\*Пример (K-Fold в Sklearn):\*\*

```python

from sklearn.model\_selection import KFold

kf = KFold(n\_splits=5, shuffle=True)

for train\_index, val\_index in kf.split(X):

X\_train, X\_val = X[train\_index], X[val\_index]

y\_train, y\_val = y[train\_index], y[val\_index]

```

#### \*\*3. Stratified K-Fold\*\*

- Сохраняет распределение классов в каждой fold (важно для несбалансированных данных).

---

## \*\*3. Контроль переобучения\*\*

\*\*Переобучение (Overfitting)\*\* — когда модель хорошо работает на обучающих данных, но плохо на новых.

### \*\*3.1. Методы борьбы\*\*

1. \*\*Ранняя остановка (Early Stopping)\*\*

- Обучение прекращается, если качество на валидации не улучшается.

- \*\*Пример в Keras\*\*:

```python

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5, # Сколько эпох ждать ухудшения

restore\_best\_weights=True

)

model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val), callbacks=[early\_stopping])

```

2. \*\*Регуляризация\*\*

- L1/L2-регуляризация (штраф за большие веса).

- Dropout (случайное "отключение" нейронов).

3. \*\*Увеличение данных (Data Augmentation)\*\*

- Для изображений: повороты, сдвиги, изменение яркости.

---

## \*\*4. Разметка данных\*\*

- \*\*Классификация\*\*: Метки должны быть целыми (0, 1) или one-hot encoded.

- \*\*Регрессия\*\*: Непрерывные значения (нормализованные).

\*\*Пример one-hot encoding\*\*:

```python

from tensorflow.keras.utils import to\_categorical

y\_train\_onehot = to\_categorical(y\_train, num\_classes=10)

```

---

## \*\*5. Нормализация данных\*\*

- \*\*Зачем?\*\* Ускоряет обучение и улучшает сходимость.

- \*\*Методы\*\*:

- Min-Max Scaling: \( X' = \frac{X - X\_{\min}}{X\_{\max} - X\_{\min}} \)

- Standard Scaling: \( X' = \frac{X - \mu}{\sigma} \)

\*\*Пример (Sklearn):\*\*

```python

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_val\_scaled = scaler.transform(X\_val) # Трансформация на тех же параметрах!

```

---

## \*\*6. Итог\*\*

1. \*\*Разделение данных\*\*: train/val/test (70/15/15).

2. \*\*Валидация\*\*: K-Fold для надежности, Early Stopping для контроля переобучения.

3. \*\*Предобработка\*\*: Нормализация и one-hot encoding.

4. \*\*Переобучение\*\*: Борьба через регуляризацию и аугментацию.

\*\*Финальный пример (Keras):\*\*

```python

model = Sequential([

Dense(64, activation='relu', input\_shape=(10,)),

Dropout(0.5),

Dense(10, activation='softmax')

])

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy']

)

history = model.fit(

X\_train\_scaled, y\_train\_onehot,

validation\_data=(X\_val\_scaled, y\_val\_onehot),

epochs=50,

batch\_size=32,

callbacks=[early\_stopping]

)

```

Вопрос № 22 Задача классификации. Формирование выборки, особенности построения архитектур НС.

### \*\*Экзаменационный вопрос: Задача классификации. Формирование выборки, особенности построения архитектур НС\*\*

---

## \*\*1. Формирование выборки для классификации\*\*

### \*\*1.1. Подготовка данных\*\*

1. \*\*Разметка данных\*\*

- Каждому объекту присваивается метка класса (например, 0 — "кошка", 1 — "собака").

- Для многоклассовой классификации используется \*\*one-hot encoding\*\*:

```python

from tensorflow.keras.utils import to\_categorical

y\_onehot = to\_categorical(y, num\_classes=3) # Преобразует [0, 1, 2] в [[1,0,0], [0,1,0], [0,0,1]]

```

2. \*\*Разделение выборки\*\*

- \*\*Обучающая (60-70%)\*\*: Для обучения модели.

- \*\*Валидационная (15-20%)\*\*: Для настройки гиперпараметров.

- \*\*Тестовая (15-20%)\*\*: Для финальной оценки.

- \*\*Пример\*\*:

```python

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=42)

```

- `stratify=y` сохраняет баланс классов.

3. \*\*Нормализация данных\*\*

- Для изображений: пиксели приводятся к диапазону [0, 1] или [-1, 1].

- Для табличных данных: StandardScaler или MinMaxScaler.

---

## \*\*2. Построение архитектуры НС для классификации\*\*

### \*\*2.1. Ключевые компоненты\*\*

1. \*\*Входной слой\*\*

- Размерность должна соответствовать данным (например, `input\_shape=(28, 28)` для изображений MNIST).

2. \*\*Скрытые слои\*\*

- \*\*Dense (полносвязные)\*\*: Для табличных данных.

- \*\*Conv2D (свёрточные)\*\*: Для изображений.

- \*\*Dropout\*\*: Для борьбы с переобучением (например, `Dropout(0.5)`).

3. \*\*Выходной слой\*\*

- \*\*Бинарная классификация\*\*: 1 нейрон с сигмоидой (`activation='sigmoid'`).

- \*\*Многоклассовая классификация\*\*: N нейронов с softmax (`activation='softmax'`), где N — число классов.

4. \*\*Функция потерь (Loss)\*\*

- Бинарная: `binary\_crossentropy`.

- Многоклассовая: `categorical\_crossentropy` (для one-hot) или `sparse\_categorical\_crossentropy` (для целых меток).

---

### \*\*2.2. Примеры архитектур\*\*

#### \*\*А. Для табличных данных (Dense-сети)\*\*

```python

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

model = Sequential([

Dense(128, activation='relu', input\_shape=(10,)), # 10 признаков на входе

Dropout(0.3),

Dense(64, activation='relu'),

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

])

model.compile(

optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy']

)

```

#### \*\*Б. Для изображений (CNN)\*\*

```python

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

Flatten(),

Dense(64, activation='relu'),

Dense(10, activation='softmax') # 10 классов для MNIST

])

model.compile(

optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy']

)

```

---

## \*\*3. Особенности обучения\*\*

### \*\*3.1. Импорт данных\*\*

- \*\*Изображения\*\*: Используйте `ImageDataGenerator` для аугментации:

```python

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.2,

horizontal\_flip=True

)

```

### \*\*3.2. Обработка дисбаланса классов\*\*

- \*\*Методы\*\*:

- Взвешенная функция потерь (`class\_weight` в `model.fit`).

- Oversampling (например, SMOTE) или undersampling.

### \*\*3.3. Контроль переобучения\*\*

- \*\*Early Stopping\*\*:

```python

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5,

restore\_best\_weights=True

)

```

- \*\*Регуляризация\*\*: L2-регуляризация в слоях:

```python

from tensorflow.keras import regularizers

Dense(64, activation='relu', kernel\_regularizer=regularizers.l2(0.01))

```

---

## \*\*4. Валидация и оценка\*\*

1. \*\*Метрики\*\*:

- Accuracy, Precision, Recall, F1-score.

- Матрица ошибок:

```python

from sklearn.metrics import confusion\_matrix

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred.argmax(axis=1))

```

2. \*\*Кросс-валидация\*\*:

```python

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(model, X, y, cv=5, scoring='accuracy')

```

---

## \*\*5. Итог\*\*

1. \*\*Данные\*\*:

- Разметка → Разделение → Нормализация.

2. \*\*Архитектура\*\*:

- Для таблиц: Dense-сети.

- Для изображений: CNN.

3. \*\*Обучение\*\*:

- Аугментация, борьба с дисбалансом, early stopping.

4. \*\*Оценка\*\*:

- Метрики качества и кросс-валидация.

\*\*Финальный пример (бинарная классификация):\*\*

```python

model = Sequential([

Dense(32, activation='relu', input\_shape=(20,)),

Dense(1, activation='sigmoid')

])

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy']

)

model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=50,

class\_weight={0: 1, 1: 2} # Учёт дисбаланса

)

```

Вопрос № 23 Переобучение нейронной сети. Причины. Способы борьбы.

### \*\*Экзаменационный вопрос: Переобучение нейронной сети. Причины. Способы борьбы\*\*

---

## \*\*1. Что такое переобучение?\*\*

\*\*Переобучение (Overfitting)\*\* — это ситуация, когда модель слишком хорошо подстраивается под обучающие данные, включая их шум и случайные колебания, и плохо обобщает на новые, ранее не виденные данные.

\*\*Признаки переобучения\*\*:

- Высокая точность на обучающей выборке, но низкая на валидационной/тестовой.

- Сильные колебания метрик (например, accuracy или loss) на валидации.

---

## \*\*2. Причины переобучения\*\*

### \*\*2.1. Основные причины\*\*

1. \*\*Слишком сложная модель\*\*

- Слишком много слоёв/нейронов для данного объёма данных.

- Пример: Глубокая сеть на маленьком датасете (например, 1 млн параметров для 1000 примеров).

2. \*\*Недостаток данных\*\*

- Модель «запоминает» примеры вместо выявления закономерностей.

3. \*\*Высокая дисперсия данных\*\*

- Наличие шума и выбросов в обучающей выборке.

4. \*\*Слишком долгое обучение\*\*

- Модель продолжает подстраиваться под шум после достижения оптимальных весов.

---

## \*\*3. Способы борьбы с переобучением\*\*

### \*\*3.1. Методы регуляризации\*\*

#### \*\*1. L1- и L2-регуляризация\*\*

- \*\*Суть\*\*: Добавление штрафа за большие веса в функцию потерь.

- \*\*L1 (Lasso)\*\*: Способствует обнулению некоторых весов.

- \*\*L2 (Ridge)\*\*: Уменьшает веса равномерно.

- \*\*Пример в Keras\*\*:

```python

from tensorflow.keras import regularizers

model.add(Dense(64,

activation='relu',

kernel\_regularizer=regularizers.l2(0.01))) # L2 с коэффициентом 0.01

```

#### \*\*2. Dropout\*\*

- \*\*Суть\*\*: Случайное «отключение» части нейронов во время обучения.

- \*\*Пример\*\*:

```python

from tensorflow.keras.layers import Dropout

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5)) # Отключает 50% нейронов

```

#### \*\*3. Batch Normalization\*\*

- \*\*Суть\*\*: Нормализация активаций в скрытых слоях.

- \*\*Пример\*\*:

```python

from tensorflow.keras.layers import BatchNormalization

model.add(Dense(128, activation='relu'))

model.add(BatchNormalization())

```

---

### \*\*3.2. Методы работы с данными\*\*

#### \*\*1. Увеличение объёма данных (Data Augmentation)\*\*

- \*\*Для изображений\*\*: Повороты, сдвиги, изменение яркости.

- \*\*Пример\*\*:

```python

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.2,

horizontal\_flip=True

)

```

#### \*\*2. Ранняя остановка (Early Stopping)\*\*

- \*\*Суть\*\*: Прекращение обучения, если качество на валидации не улучшается.

- \*\*Пример\*\*:

```python

from tensorflow.keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5, # Сколько эпох ждать ухудшения

restore\_best\_weights=True

)

model.fit(..., callbacks=[early\_stopping])

```

#### \*\*3. Кросс-валидация\*\*

- \*\*Суть\*\*: Разделение данных на K частей и поочерёдное обучение на разных подмножествах.

- \*\*Пример\*\*:

```python

from sklearn.model\_selection import KFold

kf = KFold(n\_splits=5)

for train\_index, val\_index in kf.split(X):

X\_train, X\_val = X[train\_index], X[val\_index]

y\_train, y\_val = y[train\_index], y[val\_index]

model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val))

```

---

### \*\*3.3. Архитектурные методы\*\*

#### \*\*1. Упрощение модели\*\*

- Уменьшение числа слоёв/нейронов.

- Пример:

```python

model = Sequential([

Dense(64, activation='relu', input\_shape=(100,)),

Dense(32, activation='relu'), # Вместо 128 нейронов

Dense(1, activation='sigmoid')

])

```

#### \*\*2. Использование предобученных моделей (Transfer Learning)\*\*

- \*\*Суть\*\*: Заморозка части слоёв предобученной сети (например, VGG16).

- \*\*Пример\*\*:

```python

from tensorflow.keras.applications import VGG16

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

base\_model.trainable = False # Заморозка весов

```

---

## \*\*4. Как выбрать метод?\*\*

| Ситуация | Метод борьбы |

|-----------------------------------|-----------------------------------|

| Мало данных | Data Augmentation + Transfer Learning |

| Сложная модель | Dropout + Упрощение архитектуры |

| Сильные колебания метрик | Early Stopping + Batch Norm |

| Несбалансированные классы | Взвешенная функция потерь |

---

## \*\*5. Итог\*\*

1. \*\*Причины\*\*: Сложная модель, мало данных, шум.

2. \*\*Методы борьбы\*\*:

- Регуляризация (Dropout, L1/L2).

- Работа с данными (аугментация, ранняя остановка).

- Упрощение архитектуры.

3. \*\*Важно\*\*: Всегда контролируйте метрики на валидационной выборке!

\*\*Финальный пример\*\*:

```python

model = Sequential([

Dense(128, activation='relu', input\_shape=(20,), kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

BatchNormalization(),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

history = model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=100,

batch\_size=32,

callbacks=[EarlyStopping(patience=3)],

class\_weight={0: 1, 1: 2} # Для дисбаланса классов

)

```

Вопрос № 24 Особенности построения моделей с помощью Sequential и FunctionalAPI.

### \*\*Экзаменационный вопрос: Особенности построения моделей с помощью Sequential и Functional API в Keras\*\*

---

## \*\*1. Sequential API\*\*

\*\*Sequential API\*\* — это линейный способ построения моделей, где слои добавляются последовательно один за другим.

### \*\*1.1. Основные особенности\*\*

✅ \*\*Простота\*\*: Легко использовать для простых архитектур (линейных цепочек слоёв).

❌ \*\*Ограниченность\*\*: Не поддерживает:

- Модели с несколькими входами/выходами.

- Слои с общими весами (shared layers).

- Нелинейные топологии (например, ветвления).

### \*\*1.2. Пример построения модели\*\*

```python

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

model = Sequential([

Dense(64, activation='relu', input\_shape=(784,)),

Dropout(0.5),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy')

```

### \*\*1.3. Когда использовать?\*\*

- Простые Fully Connected сети.

- CNN для классификации (например, LeNet).

- Быстрое прототипирование.

---

## \*\*2. Functional API\*\*

\*\*Functional API\*\* — гибкий способ построения сложных моделей с нелинейными связями.

### \*\*2.1. Основные особенности\*\*

✅ \*\*Гибкость\*\*: Поддерживает:

- Модели с несколькими входами/выходами.

- Общие слои (например, siamese networks).

- Сложные топологии (ветвления, residual connections).

❌ \*\*Сложность\*\*: Требует явного указания связей между слоями.

### \*\*2.2. Пример построения модели\*\*

```python

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Concatenate

# Входной слой

input\_layer = Input(shape=(784,))

# Ветвление

x = Dense(64, activation='relu')(input\_layer)

y = Dense(32, activation='relu')(input\_layer)

# Объединение ветвей

merged = Concatenate()([x, y])

# Выходной слой

output = Dense(10, activation='softmax')(merged)

# Создание модели

model = Model(inputs=input\_layer, outputs=output)

```

### \*\*2.3. Сложные архитектуры\*\*

#### \*\*А. Модель с двумя входами\*\*

```python

input1 = Input(shape=(32,))

input2 = Input(shape=(64,))

x = Dense(16, activation='relu')(input1)

y = Dense(16, activation='relu')(input2)

combined = Concatenate()([x, y])

output = Dense(1, activation='sigmoid')(combined)

model = Model(inputs=[input1, input2], outputs=output)

```

#### \*\*Б. Residual Connection (ResNet-style)\*\*

```python

input = Input(shape=(256,))

x = Dense(128, activation='relu')(input)

residual = Dense(128, activation='relu')(x)

x = Dense(128, activation='relu')(residual)

output = Dense(10, activation='softmax')(x + residual) # Сложение с residual

model = Model(inputs=input, outputs=output)

```

---

## \*\*3. Сравнение Sequential и Functional API\*\*

| Критерий | Sequential API | Functional API |

|------------------------|------------------------------------|------------------------------------|

| \*\*Простота\*\* | ✅ Очень прост | ❌ Требует описания связей |

| \*\*Гибкость\*\* | ❌ Только линейные модели | ✅ Поддержка сложных архитектур |

| \*\*Множественные входы/выходы\*\* | ❌ Не поддерживается | ✅ Полная поддержка |

| \*\*Общие слои\*\* | ❌ Нет | ✅ Есть (например, Siamese сети) |

| \*\*Отладка\*\* | ✅ Легко | ❌ Сложнее (нужно следить за shape)|

---

## \*\*4. Когда что использовать?\*\*

### \*\*4.1. Sequential API\*\*

- Однонаправленные архитектуры (CNN для MNIST, MLP).

- Быстрое прототипирование.

- Учебные примеры.

### \*\*4.2. Functional API\*\*

- Модели с \*\*несколько входами/выходами\*\* (например, мультимодальные данные).

- \*\*Сложные топологии\*\*: Residual connections, ветвления.

- \*\*Общие слои\*\* (например, для сравнения двух изображений).

- \*\*Transfer learning\*\* (заморозка части слоёв).

---

## \*\*5. Итог\*\*

1. \*\*Sequential API\*\* — для простых линейных моделей.

2. \*\*Functional API\*\* — для сложных архитектур с нелинейными связями.

3. \*\*Functional API\*\* позволяет:

- Создавать модели с несколькими входами/выходами.

- Реализовывать residual connections и ветвления.

- Использовать общие слои.

\*\*Пример перехода от Sequential к Functional API\*\*:

```python

# Sequential

model = Sequential([Dense(64, activation='relu', input\_shape=(784,))])

# Functional

input = Input(shape=(784,))

output = Dense(64, activation='relu')(input)

model = Model(inputs=input, outputs=output)

```

Вопрос № 25 Линейный слой Dense. Устройство слоя.

### \*\*Экзаменационный вопрос: Линейный слой Dense. Устройство слоя\*\*

---

## \*\*1. Что такое Dense-слой?\*\*

\*\*Dense\*\* (полносвязный слой) — это основной строительный блок нейронных сетей, где каждый нейрон слоя соединён со всеми нейронами предыдущего слоя.

\*\*Синонимы\*\*:

- Fully Connected (FC) layer.

- Linear layer (в PyTorch).

---

## \*\*2. Математическое устройство\*\*

### \*\*2.1. Формула преобразования\*\*

Для входного вектора \*\*x\*\* и весов \*\*W\*\* выход слоя вычисляется как:

\[

y = \sigma(W \cdot x + b)

\]

где:

- \( W \) — матрица весов (размерность: `units × input\_dim`),

- \( x \) — входной вектор (размерность: `input\_dim × 1`),

- \( b \) — вектор смещений (bias),

- \( \sigma \) — функция активации (может отсутствовать).

### \*\*2.2. Размерности\*\*

- Если на вход подаётся тензор формы `(batch\_size, input\_dim)`, то после Dense-слоя с `units` нейронами выход будет формы `(batch\_size, units)`.

\*\*Пример\*\*:

```python

# input\_shape = (batch\_size, 10), units = 5

dense = Dense(units=5, input\_shape=(10,))

# Выход: (batch\_size, 5)

```

---

## \*\*3. Параметры Dense-слоя\*\*

### \*\*3.1. Обязательные параметры\*\*

| Параметр | Описание | Пример |

|----------|--------------------------------------------------------------------------|------------------|

| `units` | Количество нейронов в слое (размерность выхода). | `units=64` |

### \*\*3.2. Важные аргументы\*\*

| Параметр | Описание | Пример |

|------------------|--------------------------------------------------------------------------|----------------------------|

| `activation` | Функция активации (`relu`, `sigmoid`, `softmax` и др.). | `activation='relu'` |

| `use\_bias` | Добавлять ли вектор смещений (по умолчанию `True`). | `use\_bias=False` |

| `kernel\_initializer` | Инициализация весов (`he\_normal`, `glorot\_uniform`). | `kernel\_initializer='he\_normal'` |

| `bias\_initializer` | Инициализация смещений (`zeros`, `ones`). | `bias\_initializer='zeros'` |

| `kernel\_regularizer` | Регуляризация весов (L1/L2). | `kernel\_regularizer=l2(0.01)` |

---

## \*\*4. Примеры в коде\*\*

### \*\*4.1. Создание Dense-слоя в Keras\*\*

```python

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential

model = Sequential([

Dense(units=128, activation='relu', input\_shape=(784,)), # Скрытый слой

Dense(units=10, activation='softmax') # Выходной слой

])

```

### \*\*4.2. Ручной расчёт выхода Dense-слоя\*\*

```python

import numpy as np

# Входные данные (batch\_size=2, input\_dim=3)

x = np.array([[1, 2, 3], [4, 5, 6]])

# Веса (units=2, input\_dim=3)

W = np.array([[0.1, 0.2, 0.3], [0.4, 0.5, 0.6]])

# Смещения

b = np.array([0.1, 0.2])

# Линейное преобразование

y = np.dot(x, W.T) + b # (2, 3) × (3, 2) → (2, 2)

print(y)

```

\*\*Вывод\*\*:

```

[[1.5 3.2] # Для первого примера

[3.3 7.4]] # Для второго

```

---

## \*\*5. Как работает Dense-слой в нейросети?\*\*

1. \*\*Получает входные данные\*\*: Вектор или матрица (для батча).

2. \*\*Умножает на веса\*\*: Линейная комбинация входов.

3. \*\*Добавляет смещение\*\*: Сдвигает результат.

4. \*\*Применяет активацию\*\*: Добавляет нелинейность (если задана).

\*\*Визуализация\*\*:

```

Вход (x) → [Умножение на W] → [Добавление b] → [Активация σ] → Выход (y)

```

---

## \*\*6. Особенности\*\*

1. \*\*Вычислительная сложность\*\*:

- Для входа размерности `n` и выхода `m` требуется `n × m` операций умножения.

2. \*\*Использование\*\*:

- В \*\*CNN\*\*: После свёрточных слоёв для классификации.

- В \*\*RNN\*\*: Для преобразования скрытых состояний.

3. \*\*Ограничения\*\*:

- Неэффективен для данных с пространственной структурой (например, изображений).

---

## \*\*7. Итог\*\*

1. \*\*Dense-слой\*\* выполняет линейное преобразование \( y = Wx + b \).

2. \*\*Параметры\*\*: `units`, `activation`, `use\_bias`, регуляризация.

3. \*\*Применение\*\*:

- Финал CNN (например, ResNet).

- Полносвязные сети (MLP).

\*\*Финальный пример (классификация MNIST)\*\*:

```python

model = Sequential([

Flatten(input\_shape=(28, 28)), # Преобразует изображение в вектор

Dense(128, activation='relu'),

Dense(10, activation='softmax') # 10 классов

])

```

Вопрос № 26 Сверточный слой Conv2D. Основные параметры слоя.

\*\*Сверточный слой Conv2D\*\* — один из ключевых слоёв в свёрточных нейронных сетях (CNN), предназначенный для автоматического выделения признаков из изображений или других двумерных данных.

### \*\*Основные параметры слоя Conv2D:\*\*

1. \*\*`filters` (количество фильтров)\*\*

- Определяет количество свёрточных ядер (фильтров), которые будут применяться к входным данным.

- Каждый фильтр выделяет свои особенности (например, границы, текстуры).

- Пример: `filters=32` означает, что слой создаст 32 карты признаков.

2. \*\*`kernel\_size` (размер ядра свёртки)\*\*

- Задаёт размер (высоту и ширину) окна свёртки.

- Может быть целым числом (если высота и ширина равны) или кортежем, например `(3, 3)` или `(5, 5)`.

- Чем больше ядро, тем более глобальные признаки оно может выделить, но увеличивается число параметров.

3. \*\*`strides` (шаг свёртки)\*\*

- Определяет, на сколько пикселей фильтр смещается при свёртке.

- По умолчанию `strides=(1, 1)`.

- Увеличение шага (например, `(2, 2)`) уменьшает размер выходной карты признаков.

4. \*\*`padding` (дополнение границ)\*\*

- \*\*`'valid'`\*\* — без дополнения, выходной размер уменьшается.

- \*\*`'same'`\*\* — входные данные дополняются нулями так, чтобы выходной размер совпадал с входным (если `strides=1`).

5. \*\*`activation` (функция активации)\*\*

- Обычно используется `'relu'` для нелинейности.

- Может быть `'sigmoid'`, `'tanh'` или другой функцией.

6. \*\*`input\_shape` (форма входных данных, только для первого слоя)\*\*

- Указывается для первого `Conv2D` в модели, например: `input\_shape=(28, 28, 1)` для чёрно-белых изображений 28×28.

7. \*\*`dilation\_rate` (коэффициент расширения)\*\*

- Позволяет увеличить область охвата свёртки без увеличения числа параметров (например, `dilation\_rate=2`).

8. \*\*`groups` (групповая свёртка)\*\*

- Разделяет входные и выходные каналы на группы для независимой обработки (используется, например, в MobileNet).

### \*\*Пример использования в Keras:\*\*

```python

from tensorflow.keras.layers import Conv2D

model.add(Conv2D(

filters=32,

kernel\_size=(3, 3),

strides=(1, 1),

padding='same',

activation='relu',

input\_shape=(64, 64, 3) # (высота, ширина, каналы)

))

```

Этот слой применяет 32 фильтра 3×3 с шагом 1, сохраняя размерность и добавляя нелинейность через ReLU.

### \*\*Вывод:\*\*

`Conv2D` — гибкий слой, настраиваемый под разные задачи. Основные параметры управляют количеством фильтров, размером ядра, шагом, дополнением и активацией, что позволяет эффективно извлекать пространственные признаки из данных.

Вопрос № 27 Pooling-слои.

\*\*Pooling-слои\*\* (слои подвыборки) — это важные компоненты свёрточных нейронных сетей (CNN), которые уменьшают размерность данных, сохраняя ключевую информацию. Они помогают снизить вычислительную сложность, контролировать переобучение и повысить инвариантность к небольшим сдвигам и искажениям в изображениях.

### \*\*Основные типы Pooling-слоев\*\*

1. \*\*Max Pooling (`MaxPooling2D`)\*\*

- Выбирает \*\*максимальное\*\* значение в пределах окна.

- Хорошо выделяет наиболее яркие/важные признаки.

- Пример:

```python

MaxPooling2D(pool\_size=(2, 2), strides=2) # уменьшает размер в 2 раза

```

2. \*\*Average Pooling (`AveragePooling2D`)\*\*

- Вычисляет \*\*среднее\*\* значение в окне.

- Даёт более сглаженные карты признаков.

- Пример:

```python

AveragePooling2D(pool\_size=(2, 2), strides=2)

```

3. \*\*Global Pooling\*\*

- \*\*Global Average Pooling (`GlobalAvgPool2D`)\*\* – усредняет \*\*всю\*\* карту признаков до одного значения на канал.

- \*\*Global Max Pooling (`GlobalMaxPool2D`)\*\* – берёт максимум по всей карте.

- Используется перед полносвязными слоями для замены `Flatten()`.

- Пример:

```python

GlobalAveragePooling2D() # преобразует (H, W, C) в (C,)

```

### \*\*Ключевые параметры Pooling-слоев\*\*

- \*\*`pool\_size`\*\* – размер окна (например, `(2, 2)`).

- \*\*`strides`\*\* – шаг подвыборки (если не указан, равен `pool\_size`).

- \*\*`padding`\*\* – дополнение нулями (`'valid'` или `'same'`).

### \*\*Примеры в Keras\*\*

```python

from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D, GlobalAveragePooling2D

# Max Pooling

model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='valid'))

# Average Pooling

model.add(AveragePooling2D(pool\_size=(2, 2), strides=2))

# Global Pooling (замена Flatten)

model.add(GlobalAveragePooling2D()) # выход: (batch\_size, channels)

```

### \*\*Зачем нужны Pooling-слои?\*\*

- \*\*Уменьшение размерности\*\* – снижение вычислительных затрат.

- \*\*Инвариантность к небольшим изменениям\*\* (сдвиги, повороты).

- \*\*Контроль переобучения\*\* – уменьшение числа параметров.

### \*\*Вывод\*\*

Pooling-слои – это простой, но эффективный способ сжатия информации в CNN. Max Pooling чаще используется для выделения ключевых признаков, а Average Pooling – для сглаживания. Global Pooling применяется в конце сети для перехода к полносвязным слоям.

Вопрос № 28 Tokenizer. Применение, основные параметры.

### \*\*Tokenizer: применение и основные параметры\*\*

\*\*Tokenizer\*\* (токенизатор) — это инструмент для предобработки текста, который разбивает его на отдельные элементы (токены) — слова, символы или подстроки. Применяется в задачах NLP (Natural Language Processing), таких как классификация текста, генерация, машинный перевод и др.

---

## \*\*1. Применение Tokenizer\*\*

- \*\*Разбиение текста на слова/субтокены\*\* (например, для подачи в нейросети).

- \*\*Векторизация текста\*\* (преобразование в числовые идентификаторы).

- \*\*Подготовка данных для моделей\*\* (BERT, Word2Vec, RNN, Transformers).

Пример использования в \*\*Keras\*\* и \*\*Hugging Face\*\*:

```python

from tensorflow.keras.preprocessing.text import Tokenizer

texts = ["Я изучаю NLP.", "Tokenizer полезен для обработки текста."]

# Создание и обучение токенизатора

tokenizer = Tokenizer(num\_words=100)

tokenizer.fit\_on\_texts(texts)

# Преобразование текста в последовательности чисел

sequences = tokenizer.texts\_to\_sequences(texts)

print(sequences) # Например, [[1, 2, 3], [4, 5, 6, 7, 8, 9]]

```

---

## \*\*2. Основные параметры Tokenizer (Keras)\*\*

| Параметр | Описание |

|-------------------|----------|

| \*\*`num\_words`\*\* | Максимальное количество слов в словаре (остальные заменяются на OOV-токен). |

| \*\*`filters`\*\* | Символы, которые удаляются из текста (по умолчанию: `!"#$%&()\*+,-./:;<=>?@[\\]^\_`{|}~\t\n`). |

| \*\*`lower`\*\* | Приведение текста к нижнему регистру (`True`/`False`). |

| \*\*`split`\*\* | Разделитель токенов (по умолчанию — пробел). |

| \*\*`char\_level`\*\* | Токенизация по символам (`True`), а не по словам (`False`). |

| \*\*`oov\_token`\*\* | Токен для неизвестных слов (например, `"<OOV>"`). |

Пример с параметрами:

```python

tokenizer = Tokenizer(

num\_words=1000,

filters='!"#$%&()\*+,-./:;<=>?@[\\]^\_`{|}~\t\n',

lower=True,

split=' ',

oov\_token="<UNK>"

)

```

---

## \*\*3. Дополнительные методы Tokenizer\*\*

- \*\*`fit\_on\_texts(texts)`\*\* — создаёт словарь на основе текста.

- \*\*`texts\_to\_sequences(texts)`\*\* — преобразует текст в последовательность чисел.

- \*\*`sequences\_to\_texts(sequences)`\*\* — обратное преобразование чисел в текст.

- \*\*`word\_index`\*\* — словарь вида `{"слово": id}`.

Пример:

```python

print(tokenizer.word\_index) # {'я': 1, 'изучаю': 2, 'nlp': 3, ...}

```

---

## \*\*4. Tokenizer в Hugging Face (Transformers)\*\*

В библиотеке `transformers` используются более сложные токенизаторы (например, для BERT, GPT):

```python

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

text = "Hello, NLP!"

tokens = tokenizer(text, padding=True, truncation=True, return\_tensors="pt")

```

\*\*Параметры:\*\*

- `padding` — дополнение до максимальной длины.

- `truncation` — обрезка длинных текстов.

- `return\_tensors` — формат вывода (`"pt"` для PyTorch, `"tf"` для TensorFlow).

---

## \*\*Вывод\*\*

- \*\*Tokenizer\*\* разбивает текст на токены и преобразует их в числа.

- В \*\*Keras\*\* он используется для простых моделей, в \*\*Hugging Face\*\* — для современных NLP-архитектур.

- Основные параметры: `num\_words`, `filters`, `oov\_token`, `lower`.

- Применяется во всех задачах обработки естественного языка (NLP).

Пример экзаменационного ответа:

> \*"Tokenizer — это инструмент для разбиения текста на токены (слова или символы) и их числовой векторзации. Основные параметры включают `num\_words`, `filters`, `lower` и `oov\_token`. Используется в нейросетях для обработки текстовых данных, например, в Keras и библиотеке Hugging Face Transformers."\*

Вопрос № 29 Параметризация текста BagOfWords.

### \*\*Bag of Words (BoW) — параметризация текста\*\*

\*\*Bag of Words (BoW, "мешок слов")\*\* — это простой, но мощный метод представления текста в виде числового вектора, где каждое слово учитывается как независимая единица без учёта порядка слов.

---

## \*\*1. Суть метода\*\*

- Текст разбивается на слова (токены).

- Создаётся словарь уникальных слов из всего корпуса текстов.

- Каждый документ кодируется вектором, где:

- \*\*Размерность\*\* = размер словаря.

- \*\*Значения\*\* = частота (или наличие) слова в документе.

\*\*Пример:\*\*

```python

Текст 1: "кот ест рыбу"

Текст 2: "собака ест мясо"

Словарь: {'кот':0, 'ест':1, 'рыбу':2, 'собака':3, 'мясо':4}

Векторизация:

Текст 1 → [1, 1, 1, 0, 0]

Текст 2 → [0, 1, 0, 1, 1]

```

---

## \*\*2. Основные параметры и варианты BoW\*\*

### \*\*1. Бинарный BoW (Binary Bag of Words)\*\*

- `1` — слово есть в документе, `0` — отсутствует.

- Упрощённый вариант, подходит для классификации.

### \*\*2. Частотный BoW (Count-Based BoW)\*\*

- Значения = количество вхождений слова.

- Пример: `"кот ест рыбу, кот спит"` → `[2, 1, 1, 0, 0]`.

### \*\*3. TF-IDF (Term Frequency-Inverse Document Frequency)\*\*

- Учитывает \*\*важность слова\*\* в документе относительно всего корпуса:

- \*\*TF (Term Frequency)\*\* — частота слова в документе.

- \*\*IDF (Inverse Document Frequency)\*\* — логарифм обратной частоты слова в корпусе.

- Формула:

\[

\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log\left(\frac{N}{\text{DF}(t)}\right)

\]

где:

- \(N\) — общее число документов,

- \(\text{DF}(t)\) — число документов, содержащих слово \(t\).

---

## \*\*3. Параметры реализации (на примере `sklearn`)\*\*

В библиотеке `scikit-learn` BoW реализуется через:

- \*\*`CountVectorizer`\*\* (частотный BoW),

- \*\*`TfidfVectorizer`\*\* (TF-IDF).

### \*\*Основные параметры:\*\*

| Параметр | Описание |

|-------------------------|----------|

| \*\*`max\_features`\*\* | Максимальное количество слов в словаре. |

| \*\*`stop\_words`\*\* | Удаление стоп-слов (например, `{"и", "в", "на"}`). |

| \*\*`ngram\_range`\*\* | Учитывать N-граммы (например, `(1, 2)` для слов и биграмм). |

| \*\*`min\_df` / `max\_df`\*\* | Игнорировать слова, встречающиеся реже/чаще указанного порога. |

| \*\*`binary`\*\* | Бинарный режим (`True`/`False`). |

### \*\*Пример кода:\*\*

```python

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

corpus = ["кот ест рыбу", "собака ест мясо"]

# Частотный BoW

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(corpus)

print(vectorizer.get\_feature\_names\_out()) # ['кот', 'мясо', 'рыбу', 'собака', 'ест']

print(X.toarray()) # [[1, 0, 1, 0, 1], [0, 1, 0, 1, 1]]

# TF-IDF

tfidf = TfidfVectorizer()

X\_tfidf = tfidf.fit\_transform(corpus)

```

---

## \*\*4. Плюсы и минусы BoW\*\*

✅ \*\*Плюсы:\*\*

- Простота реализации и интерпретации.

- Эффективен для задач классификации (спам, тональность).

- Работает с любыми языками.

❌ \*\*Минусы:\*\*

- Игнорирует порядок слов и контекст ("плохой" и "не плохой" будут одинаковы).

- Разреженность данных (векторы с большим количеством нулей).

- Плохо работает с омонимами и синонимами.

---

## \*\*5. Вывод\*\*

\*\*Bag of Words\*\* — это базовый метод векторизации текста, который представляет документы в виде числовых векторов на основе частоты слов.

- \*\*Варианты:\*\* бинарный, частотный, TF-IDF.

- \*\*Параметры:\*\* `max\_features`, `stop\_words`, `ngram\_range`.

- \*\*Применение:\*\* классификация текста, поиск документов, простые NLP-модели.

\*\*Пример ответа на экзамене:\*\*

> \*"Bag of Words — метод параметризации текста, при котором документ представляется вектором частот слов из словаря. Основные варианты: бинарный, частотный и TF-IDF. Параметры включают `max\_features`, `stop\_words` и `ngram\_range`. Плюсы — простота, минусы — потеря контекста и разреженность данных."\*

Вопрос № 30 Embedding-слой. Основные параметры слоя.

### \*\*Embedding-слой: назначение и основные параметры\*\*

\*\*Embedding-слой\*\* (слой векторного представления) — это ключевой компонент нейросетей для обработки категориальных данных (например, текстов), который преобразует дискретные токены (слова, символы, ID) в плотные числовые векторы.

---

## \*\*1. Назначение Embedding-слоя\*\*

- Переводит \*\*токены\*\* (слова, категории) в \*\*векторы фиксированной размерности\*\*.

- Учитывает семантические связи между объектами (например, "король" ~ "королева" + "женщина" – "мужчина").

- Используется в:

- NLP (обработка текста: Word2Vec, BERT, GPT).

- Рекомендательных системах (векторизация пользователей/товаров).

- Работе с категориальными признаками в табличных данных.

---

## \*\*2. Основные параметры Embedding-слоя\*\*

В библиотеках \*\*Keras/TensorFlow\*\* и \*\*PyTorch\*\* параметры схожи:

| Параметр | Описание |

|-------------------------|----------|

| \*\*`input\_dim`\*\* | Размер словаря (количество уникальных токенов). |

| \*\*`output\_dim`\*\* | Размерность векторного пространства (например, 100, 300). |

| \*\*`input\_length`\*\* | Длина входной последовательности (для текста — число слов). |

| \*\*`mask\_zero`\*\* | Игнорировать нулевые токены (например, паддинг). |

| \*\*`trainable`\*\* | Можно ли обучать эмбеддинги (`True`/`False`). |

| \*\*`weights`\*\* | Предобученные веса (например, из Word2Vec). |

---

## \*\*3. Примеры реализации\*\*

### \*\*Keras/TensorFlow\*\*

```python

from tensorflow.keras.layers import Embedding

# Слой для текста (словарь из 10000 слов, каждое слово → вектор размерности 128)

embedding\_layer = Embedding(

input\_dim=10000, # Размер словаря

output\_dim=128, # Размерность эмбеддингов

input\_length=50, # Максимальная длина предложения

mask\_zero=True, # Игнорировать нулевые токены (паддинг)

trainable=True # Веса будут обновляться при обучении

)

```

### \*\*PyTorch\*\*

```python

import torch.nn as nn

embedding = nn.Embedding(

num\_embeddings=10000, # Аналог input\_dim

embedding\_dim=128, # Аналог output\_dim

padding\_idx=0 # Игнорировать токен 0 (паддинг)

)

```

---

## \*\*4. Как работает Embedding-слой?\*\*

1. На вход подаются \*\*целочисленные индексы\*\* (например, `[3, 1, 0, 4]`).

2. Каждый индекс заменяется на соответствующий вектор из матрицы `(input\_dim × output\_dim)`.

3. На выходе получается тензор формы `(batch\_size, input\_length, output\_dim)`.

\*\*Пример:\*\*

```python

# Вход: 2 предложения по 3 слова (словарь = 10, размерность = 4)

input\_data = [[1, 2, 3], [4, 5, 0]] # 0 — паддинг

embedding\_layer(input\_data) # Выход: (2, 3, 4)

```

---

## \*\*5. Предобученные эмбеддинги\*\*

Embedding-слой можно инициализировать готовыми векторами (например, \*\*GloVe\*\* или \*\*Word2Vec\*\*):

```python

import numpy as np

# Загрузка предобученных весов (например, GloVe)

embedding\_matrix = np.load("glove\_vectors.npy") # Матрица (input\_dim × output\_dim)

embedding\_layer = Embedding(

input\_dim=10000,

output\_dim=300,

weights=[embedding\_matrix],

trainable=False # Заморозить веса

)

```

---

## \*\*6. Плюсы и минусы\*\*

✅ \*\*Плюсы:\*\*

- Преобразует категории в осмысленные векторы.

- Уменьшает размерность данных (по сравнению с one-hot encoding).

- Учитывает семантику (если веса обучаются на большом корпусе).

❌ \*\*Минусы:\*\*

- Требует большого объема данных для обучения.

- Фиксированный словарь (`input\_dim` должен быть задан заранее).

---

## \*\*7. Вывод\*\*

\*\*Embedding-слой\*\* — это:

- Матрица перевода токенов в векторы.

- Основные параметры: `input\_dim`, `output\_dim`, `input\_length`, `trainable`.

- Применяется в NLP, рекомендательных системах и других задачах с категориальными данными.

\*\*Пример ответа на экзамене:\*\*

> \*"Embedding-слой преобразует дискретные токены (слова, ID) в плотные векторы фиксированной размерности. Ключевые параметры: input\_dim (размер словаря), output\_dim (размерность векторов), input\_length (длина последовательности). Используется в нейросетях для работы с текстом и категориальными признаками."\*

Вопрос № 31 Задач регрессии. Примеры. Подготовка данных, принцип построения архитектур НС для решения задачи.

### \*\*Задачи регрессии в нейронных сетях\*\*

\*\*Регрессия\*\* — это задача прогнозирования непрерывной числовой величины (например, цены, температуры, спроса). В отличие от классификации, где выход — это дискретный класс, в регрессии результат — это вещественное число.

---

## \*\*1. Примеры задач регрессии\*\*

1. \*\*Прогнозирование цен\*\*

- Предсказание стоимости недвижимости на основе характеристик (площадь, район, этаж).

2. \*\*Медицина\*\*

- Прогноз уровня глюкозы в крови у пациентов.

3. \*\*Техника\*\*

- Оценка остаточного ресурса двигателя по данным датчиков.

4. \*\*Экономика\*\*

- Предсказание ВВП страны на основе макроэкономических показателей.

5. \*\*Продажи\*\*

- Прогноз спроса на товар в зависимости от сезона и рекламных активностей.

---

## \*\*2. Подготовка данных\*\*

### \*\*1. Особенности данных для регрессии\*\*

- \*\*Целевая переменная (y)\*\* — вещественное число.

- \*\*Признаки (X)\*\* — могут быть числовыми, категориальными, временными рядами.

### \*\*2. Предобработка данных\*\*

- \*\*Нормализация/стандартизация\*\* (обязательно для нейросетей):

```python

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

```

- \*\*Кодирование категориальных признаков\*\* (One-Hot, Label Encoding).

- \*\*Заполнение пропусков\*\* (средним, медианой, предсказанием).

- \*\*Разделение на train/test\*\* (обычно 80/20 или 70/30).

---

## \*\*3. Принципы построения архитектуры НС\*\*

### \*\*1. Базовые архитектуры\*\*

- \*\*Полносвязная сеть (Dense)\*\* — для табличных данных.

- \*\*Свёрточная сеть (CNN)\*\* — если данные имеют пространственную структуру (например, изображения с прогнозируемым параметром).

- \*\*Рекуррентная сеть (RNN/LSTM)\*\* — для временных рядов (например, прогноз температуры по истории).

### \*\*2. Ключевые параметры архитектуры\*\*

- \*\*Входной слой\*\*:

```python

tf.keras.layers.Input(shape=(n\_features,))

```

- \*\*Скрытые слои\*\*:

- Обычно \*\*Dense\*\* с активацией `ReLU` или `LeakyReLU`.

- Глубина: 1–5 слоёв (зависит от сложности задачи).

- Число нейронов: 32–512 (подбирается экспериментально).

- \*\*Выходной слой\*\*:

```python

tf.keras.layers.Dense(1, activation='linear') # 1 выход для регрессии

```

- \*\*Регуляризация\*\*:

- Dropout, BatchNorm, L1/L2-регуляризация.

### \*\*3. Функция потерь и метрики\*\*

- \*\*Loss\*\*:

- \*\*MSE (Mean Squared Error)\*\* — стандартный выбор.

- \*\*MAE (Mean Absolute Error)\*\* — если выбросы важны.

- \*\*Huber Loss\*\* — компромисс между MSE и MAE.

- \*\*Метрики\*\*:

- \*\*RMSE\*\* (корень из MSE), \*\*R²\*\* (коэффициент детерминации).

### \*\*4. Пример архитектуры на Keras\*\*

```python

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(10,)), # 10 признаков

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(32, activation='relu'),

tf.keras.layers.Dense(1) # Выход регрессии

])

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

history = model.fit(X\_train, y\_train, epochs=50, validation\_split=0.2)

```

---

## \*\*4. Особенности обучения\*\*

- \*\*Оптимизатор\*\*: Adam, SGD, RMSprop.

- \*\*Early Stopping\*\*: Остановка при переобучении.

```python

callback = tf.keras.callbacks.EarlyStopping(patience=5)

```

- \*\*Подбор learning rate\*\*:

```python

tf.keras.optimizers.Adam(learning\_rate=0.001)

```

---

## \*\*5. Валидация и интерпретация\*\*

- \*\*Графики обучения\*\*:

- Следить за `loss` и `val\_loss`.

- \*\*Предсказания vs Реальность\*\*:

```python

plt.scatter(y\_test, model.predict(X\_test))

```

- \*\*Feature Importance\*\*: Анализ значимости признаков (SHAP, Permutation Importance).

---

## \*\*6. Вывод\*\*

1. \*\*Задачи регрессии\*\* предсказывают числовые значения.

2. \*\*Данные\*\* требуют нормализации и обработки пропусков.

3. \*\*Архитектура НС\*\*:

- Входной слой = число признаков,

- Выходной слой = 1 нейрон с линейной активацией,

- Loss = MSE/MAE.

4. \*\*Обучение\*\*: контроль переобучения, подбор LR.

\*\*Пример ответа на экзамене:\*\*

> \*"Задача регрессии в нейронных сетях — это прогнозирование непрерывной величины (например, цены или температуры). Данные требуют нормализации, а архитектура обычно включает полносвязные слои с активацией ReLU и один выходной нейрон с линейной функцией. Функция потерь — MSE или MAE. Для валидации используются метрики RMSE и R²."\*

Вопрос № 32 Временный ряды. Инструмент TimeSeriesGenerator. Особенности формирования выборки.

### \*\*Временные ряды и TimeSeriesGenerator\*\*

\*\*Временные ряды\*\* — это данные, измеренные последовательно во времени (например, курс акций, температура, продажи). Для их обработки в нейронных сетях используется \*\*TimeSeriesGenerator\*\*, который преобразует временные данные в формат, пригодный для обучения.

---

## \*\*1. Особенности временных рядов\*\*

- \*\*Зависимость от времени\*\*: Каждое значение связано с предыдущими.

- \*\*Тренды и сезонность\*\*: Данные могут иметь долгосрочные тренды и периодические колебания.

- \*\*Нестационарность\*\*: Статистические свойства (среднее, дисперсия) могут меняться со временем.

---

## \*\*2. Инструмент `TimeseriesGenerator` (Keras)\*\*

Класс `TimeseriesGenerator` автоматически разбивает временной ряд на \*\*последовательности "окон"\*\* для обучения RNN/LSTM.

### \*\*Параметры генератора:\*\*

| Параметр | Описание |

|-------------------|----------|

| \*\*`data`\*\* | Исходные данные (массив `[x1, x2, ..., xn]`). |

| \*\*`targets`\*\* | Целевые значения (обычно сдвинутые на 1 шаг вперёд). |

| \*\*`length`\*\* | Размер окна (число предыдущих шагов для прогноза). |

| \*\*`sampling\_rate`\*\* | Частота выборки (например, `1` — каждый шаг, `2` — через шаг). |

| \*\*`stride`\*\* | Шаг перемещения окна (по умолчанию `1`). |

| \*\*`batch\_size`\*\* | Размер батча для обучения. |

### \*\*Пример создания генератора:\*\*

```python

from keras.preprocessing.sequence import TimeseriesGenerator

import numpy as np

data = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

targets = data[1:] # Предсказываем следующий элемент

generator = TimeseriesGenerator(

data=data,

targets=targets,

length=3, # Используем 3 предыдущих значения

batch\_size=2 # Размер батча

)

# Получаем первый батч

X\_batch, y\_batch = generator[0]

print(X\_batch) # [[1, 2, 3], [2, 3, 4]]

print(y\_batch) # [4, 5]

```

---

## \*\*3. Формирование выборки\*\*

### \*\*1. Разметка данных\*\*

- Для каждого момента времени `t` входом являются предыдущие `n` значений (`t-n, ..., t-1`), а выходом — значение в момент `t`.

- Пример для `length=3`:

```

X: [1, 2, 3] → y: 4

X: [2, 3, 4] → y: 5

```

### \*\*2. Разделение на train/test\*\*

- \*\*Важно:\*\* Нельзя перемешивать данные!

- Обычно берут \*\*первые 80%\*\* для обучения, \*\*последние 20%\*\* для теста.

### \*\*3. Нормализация данных\*\*

- \*\*MinMaxScaler\*\* или \*\*StandardScaler\*\* применяется ко всему ряду перед разбиением.

```python

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

data\_scaled = scaler.fit\_transform(data.reshape(-1, 1))

```

---

## \*\*4. Пример архитектуры LSTM для временных рядов\*\*

```python

from keras.models import Sequential

from keras.layers import LSTM, Dense

model = Sequential([

LSTM(50, activation='relu', input\_shape=(3, 1)), # 3 временных шага, 1 признак

Dense(1) # Выход регрессии

])

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

model.fit(generator, epochs=100)

```

---

## \*\*5. Особенности работы с TimeSeriesGenerator\*\*

- \*\*Наложение окон\*\*: При `stride=1` окна перекрываются, что увеличивает объём данных.

- \*\*Прогноз на несколько шагов вперёд\*\*:

- Можно модифицировать `targets`, чтобы предсказывать не `t+1`, а `t+k`.

- \*\*Многомерные ряды\*\*: Если данных несколько (например, температура + влажность), `data` должен быть 2D-массивом.

---

## \*\*6. Альтернативы\*\*

- \*\*`tf.keras.utils.timeseries\_dataset\_from\_array`\*\* (в TensorFlow 2.6+).

- \*\*Ручная реализация\*\* через `np.lib.stride\_tricks.sliding\_window\_view`.

---

## \*\*7. Вывод\*\*

1. \*\*TimeSeriesGenerator\*\* преобразует временной ряд в последовательность окон для обучения.

2. \*\*Ключевые параметры\*\*: `length` (размер окна), `batch\_size`, `stride`.

3. \*\*Данные\*\* должны быть нормализованы и разделены без перемешивания.

4. \*\*Архитектура НС\*\*: Обычно LSTM или GRU с выходным Dense-слоем.

\*\*Пример ответа на экзамене:\*\*

> \*"TimeSeriesGenerator в Keras используется для подготовки временных рядов к обучению нейросетей. Он создаёт скользящие окна из `length` предыдущих значений, чтобы предсказать следующее. Данные должны быть нормализованы и разделены на train/test без перемешивания. Типовая архитектура — LSTM с полносвязным выходным слоем."\*

Вопрос № 33 Аугментация изображений. Инструмент ImageDataGenerator.

### \*\*Аугментация изображений и ImageDataGenerator\*\*

\*\*Аугментация изображений\*\* — это метод искусственного увеличения обучающей выборки за счёт применения случайных преобразований к исходным изображениям. Это помогает улучшить обобщающую способность модели и избежать переобучения.

---

## \*\*1. Основные виды аугментации\*\*

- \*\*Геометрические преобразования\*\*:

- Поворот (`rotation\_range`)

- Сдвиг (`width\_shift\_range`, `height\_shift\_range`)

- Отражение (`horizontal\_flip`, `vertical\_flip`)

- Масштабирование (`zoom\_range`)

- \*\*Яркостные преобразования\*\*:

- Изменение яркости (`brightness\_range`)

- Нормализация (`rescale`)

- \*\*Искажения\*\*:

- Обрезка (`random\_crop`)

- Заполнение (`fill\_mode` — "nearest", "reflect")

---

## \*\*2. Инструмент `ImageDataGenerator` (Keras)\*\*

Класс `ImageDataGenerator` автоматически применяет аугментацию в реальном времени во время обучения модели.

### \*\*Основные параметры:\*\*

| Параметр | Описание | Пример значения |

|-------------------------|----------|------------------|

| \*\*`rotation\_range`\*\* | Поворот на угол (градусы) | `30` |

| \*\*`width\_shift\_range`\*\* | Сдвиг по ширине (доля от ширины) | `0.2` |

| \*\*`height\_shift\_range`\*\*| Сдвиг по высоте (доля от высоты) | `0.2` |

| \*\*`shear\_range`\*\* | Сдвиг (деформация) | `0.2` |

| \*\*`zoom\_range`\*\* | Масштабирование | `[0.8, 1.2]` |

| \*\*`horizontal\_flip`\*\* | Горизонтальное отражение | `True` |

| \*\*`vertical\_flip`\*\* | Вертикальное отражение | `True` |

| \*\*`brightness\_range`\*\* | Изменение яркости | `[0.5, 1.5]` |

| \*\*`rescale`\*\* | Нормализация пикселей (например, `/255`) | `1./255` |

| \*\*`fill\_mode`\*\* | Заполнение краёв | `"nearest"` |

### \*\*Пример создания генератора:\*\*

```python

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=30,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

vertical\_flip=False,

brightness\_range=[0.8, 1.2],

rescale=1./255

)

```

---

## \*\*3. Использование генератора в обучении\*\*

### \*\*1. Загрузка данных\*\*

Генератор может работать с данными:

- В памяти (NumPy-массивы)

- Из папок (`flow\_from\_directory`)

```python

train\_generator = datagen.flow\_from\_directory(

'data/train',

target\_size=(150, 150), # Размер изображений

batch\_size=32,

class\_mode='binary' # Для бинарной классификации

)

```

### \*\*2. Обучение модели\*\*

```python

model.fit(

train\_generator,

steps\_per\_epoch=len(train\_generator),

epochs=50

)

```

---

## \*\*4. Визуализация аугментации\*\*

Чтобы увидеть, как преобразуются изображения:

```python

import matplotlib.pyplot as plt

# Берем одно изображение для примера

x, y = next(train\_generator)

# Выводим первые 5 аугментированных изображений

plt.figure(figsize=(10, 5))

for i in range(5):

plt.subplot(1, 5, i+1)

plt.imshow(x[i])

plt.axis('off')

plt.show()

```

---

## \*\*5. Особенности аугментации\*\*

- \*\*Режим заполнения (`fill\_mode`)\*\* — важно для краёв изображения:

- `"nearest"` — повтор крайних пикселей.

- `"reflect"` — зеркальное отражение.

- `"constant"` — заполнение нулями.

- \*\*Аугментация только при обучении\*\* — на валидации и тесте преобразования не применяются.

- \*\*Совмещение с другими методами\*\* — можно добавлять случайный шум, размытие и т.д.

---

## \*\*6. Альтернативы\*\*

- \*\*Albumentations\*\* — более мощная библиотека для аугментации.

- \*\*Torchvision (PyTorch)\*\* — `transforms.Compose([...])`.

---

## \*\*7. Вывод\*\*

1. \*\*Аугментация\*\* увеличивает разнообразие данных и улучшает обобщение модели.

2. \*\*`ImageDataGenerator`\*\* — удобный инструмент в Keras для аугментации "на лету".

3. \*\*Основные преобразования\*\*: поворот, сдвиг, отражение, масштабирование.

4. \*\*Важно\*\*: Аугментация применяется только к обучающим данным.

\*\*Пример ответа на экзамене:\*\*

> \*"Аугментация изображений — это метод увеличения обучающей выборки за счёт случайных преобразований (поворот, сдвиг, отражение). В Keras для этого используется `ImageDataGenerator`, который применяет аугментацию в реальном времени. Параметры включают `rotation\_range`, `horizontal\_flip`, `zoom\_range` и др. Аугментация помогает избежать переобучения и улучшает работу модели."\*

Вопрос № 34 Устройство автокодировщика. Области применения и недостатки подхода.

### \*\*Устройство автокодировщика (Autoencoder)\*\*

\*\*Автокодировщик\*\* — это нейронная сеть, которая учится сжимать входные данные в компактное представление (кодировку), а затем восстанавливать их с минимальными потерями.

---

## \*\*1. Архитектура автокодировщика\*\*

Автокодировщик состоит из двух основных частей:

1. \*\*Энкодер (Encoder)\*\*

- Сжимает входные данные в \*\*скрытое представление\*\* (латентный вектор).

- Пример архитектуры:

```python

encoder = Sequential([

Dense(128, activation='relu', input\_shape=(input\_dim,)),

Dense(64, activation='relu'),

Dense(latent\_dim) # Например, latent\_dim = 32

])

```

2. \*\*Декодер (Decoder)\*\*

- Восстанавливает данные из латентного вектора.

- Пример архитектуры:

```python

decoder = Sequential([

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

Dense(128, activation='relu'),

Dense(input\_dim, activation='sigmoid') # Восстановление исходного размера

])

```

3. \*\*Объединённая модель\*\*

```python

autoencoder = Model(inputs=encoder.inputs, outputs=decoder(encoder.outputs))

autoencoder.compile(optimizer='adam', loss='mse')

```

---

## \*\*2. Области применения\*\*

### \*\*1. Сжатие данных (Dimensionality Reduction)\*\*

- Альтернатива PCA, но с нелинейными преобразованиями.

- Пример: уменьшение размерности изображений.

### \*\*2. Шумоподавление (Denoising Autoencoder)\*\*

- Обучается на зашумленных данных, чтобы восстанавливать чистые.

```python

noisy\_data = original\_data + noise

autoencoder.fit(noisy\_data, original\_data)

```

### \*\*3. Генерация данных (Variational Autoencoder, VAE)\*\*

- Генерирует новые объекты, похожие на обучающую выборку.

### \*\*4. Обнаружение аномалий (Anomaly Detection)\*\*

- Если ошибка восстановления велика → объект аномальный.

### \*\*5. Предобучение моделей (Pre-training)\*\*

- Энкодер можно использовать как инициализацию для других задач.

---

## \*\*3. Недостатки подхода\*\*

1. \*\*Качество сжатия\*\*

- Обычно хуже, чем у специализированных алгоритмов (JPEG, PCA).

2. \*\*Интерпретируемость\*\*

- Латентное пространство сложно анализировать (в отличие от PCA).

3. \*\*Неэффективность для простых данных\*\*

- Для табличных данных с линейными зависимостями PCA может быть лучше.

4. \*\*Баланс между сжатием и качеством\*\*

- Слишком маленький `latent\_dim` → большие потери.

- Слишком большой → нет сжатия.

5. \*\*Вычислительная сложность\*\*

- Требует больших данных и времени для обучения.

---

## \*\*4. Примеры кода\*\*

### \*\*Denoising Autoencoder в Keras\*\*

```python

from keras.layers import Input, Dense

from keras.models import Model

# Энкодер

input\_img = Input(shape=(784,))

encoded = Dense(128, activation='relu')(input\_img)

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

latent = Dense(32, activation='relu')(encoded)

# Декодер

decoded = Dense(64, activation='relu')(latent)

decoded = Dense(128, activation='relu')(decoded)

decoded = Dense(784, activation='sigmoid')(decoded)

# Модель

autoencoder = Model(input\_img, decoded)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Обучение на зашумленных данных

autoencoder.fit(noisy\_train, clean\_train, epochs=50)

```

### \*\*VAE (Variational Autoencoder)\*\*

```python

# Вариационный автокодировщик добавляет стохастичность

z\_mean = Dense(latent\_dim)(encoded)

z\_log\_var = Dense(latent\_dim)(encoded)

z = Lambda(sampling)([z\_mean, z\_log\_var]) # Случайный сэмплинг

```

---

## \*\*5. Вывод\*\*

✅ \*\*Плюсы автокодировщиков\*\*:

- Универсальность (сжатие, шумоподавление, генерация).

- Возможность работать с изображениями, текстами, звуком.

❌ \*\*Минусы\*\*:

- Не всегда лучше классических методов (PCA, JPEG).

- Сложность интерпретации латентного пространства.

\*\*Пример ответа на экзамене:\*\*

> \*"Автокодировщик состоит из энкодера (сжимает данные) и декодера (восстанавливает). Применяется для сжатия, шумоподавления и генерации данных. Недостатки: низкая интерпретируемость латентного пространства и неэффективность для простых линейных зависимостей. Пример реализации — Denoising Autoencoder или VAE."\*

Вопрос № 35 Способы сохранения/загрузки модели нейронной сети.

### \*\*Способы сохранения и загрузки модели нейронной сети\*\*

Сохранение и загрузка моделей — важный этап в машинном обучении, позволяющий:

- \*\*Продолжить обучение\*\* с последнего состояния.

- \*\*Использовать модель\*\* без переобучения.

- \*\*Делиться моделью\*\* с другими разработчиками.

---

## \*\*1. Сохранение всей модели (архитектура + веса + оптимизатор)\*\*

### \*\*Keras / TensorFlow\*\*

```python

# Сохранение

model.save("full\_model.keras") # или .h5 в TF < 2.16

# Загрузка

loaded\_model = tf.keras.models.load\_model("full\_model.keras")

```

### \*\*PyTorch\*\*

```python

# Сохранение

torch.save(model, "full\_model.pth")

# Загрузка

loaded\_model = torch.load("full\_model.pth")

```

\*\*Плюсы\*\*:

- Сохраняет всё: архитектуру, веса, состояние оптимизатора.

\*\*Минусы\*\*:

- Может быть несовместимо между версиями фреймворков.

---

## \*\*2. Сохранение только весов\*\*

### \*\*Keras / TensorFlow\*\*

```python

# Сохранение весов

model.save\_weights("weights.ckpt") # или .h5

# Загрузка (требуется создание модели перед загрузкой!)

model.load\_weights("weights.ckpt")

```

### \*\*PyTorch\*\*

```python

# Сохранение

torch.save(model.state\_dict(), "weights.pth")

# Загрузка

model.load\_state\_dict(torch.load("weights.pth"))

```

\*\*Плюсы\*\*:

- Меньший размер файла.

- Совместимость между разными версиями (если архитектура не менялась).

\*\*Минусы\*\*:

- Требуется идентичная архитектура при загрузке.

---

## \*\*3. Сохранение архитектуры (без весов)\*\*

### \*\*Keras / TensorFlow\*\*

```python

# Сохранение архитектуры в JSON

json\_config = model.to\_json()

with open("model\_config.json", "w") as f:

f.write(json\_config)

# Загрузка

with open("model\_config.json", "r") as f:

loaded\_model = tf.keras.models.model\_from\_json(f.read())

```

### \*\*PyTorch\*\*

Архитектуру можно сохранить как код Python или через `torch.jit.script`.

\*\*Плюсы\*\*:

- Позволяет воссоздать модель без весов.

\*\*Минусы\*\*:

- Веса нужно сохранять отдельно.

---

## \*\*4. ONNX (кроссплатформенный формат)\*\*

```python

# TensorFlow → ONNX

import tf2onnx

tf2onnx.convert.from\_keras(model, output\_path="model.onnx")

# PyTorch → ONNX

torch.onnx.export(model, dummy\_input, "model.onnx")

```

\*\*Плюсы\*\*:

- Совместимость между фреймворками (TensorFlow, PyTorch, ONNX Runtime).

\*\*Минусы\*\*:

- Не все операции поддерживаются.

---

## \*\*5. Сохранение в TensorFlow SavedModel\*\*

```python

# Сохранение

tf.saved\_model.save(model, "saved\_model\_dir")

# Загрузка

loaded\_model = tf.saved\_model.load("saved\_model\_dir")

```

\*\*Особенности\*\*:

- Подходит для продакшена и развёртывания в TF Serving.

---

## \*\*6. Дополнительные методы\*\*

### \*\*Pickle (только для PyTorch)\*\*

```python

import pickle

# Сохранение

with open("model.pkl", "wb") as f:

pickle.dump(model, f)

# Загрузка

with open("model.pkl", "rb") as f:

loaded\_model = pickle.load(f)

```

\*\*Осторожно\*\*: Может быть небезопасным из-за исполняемого кода.

---

## \*\*7. Какой способ выбрать?\*\*

| Способ | Когда использовать? |

|----------------------------|-----------------------------------------|

| \*\*Полная модель (.keras/.h5/.pth)\*\* | Для быстрого сохранения/загрузки. |

| \*\*Только веса (.ckpt/.pth)\*\* | Для передачи весов между идентичными моделями. |

| \*\*ONNX\*\* | Для кроссплатформенного использования. |

| \*\*SavedModel\*\* | Для развёртывания в продакшене (TensorFlow). |

---

## \*\*8. Пример ответа на экзамене\*\*

> \*"Способы сохранения моделей включают: 1) полное сохранение (архитектура + веса + оптимизатор) через `model.save()`, 2) сохранение только весов (`save\_weights()`), 3) экспорт в ONNX для кроссплатформенности. В PyTorch используют `state\_dict()` и `torch.save()`. Для продакшена в TensorFlow применяют SavedModel, а в PyTorch — TorchScript."\*

Вопрос № 36 Архитектура Вариационных автокодировщиков. Ошибка Кульбака-Лейблера.

### \*\*Архитектура вариационных автокодировщиков (VAE) и ошибка Кульбака-Лейблера\*\*

\*\*Вариационный автокодировщик (VAE, Variational Autoencoder)\*\* — это генеративная модель, которая учится сжимать данные в вероятностное латентное пространство и генерировать новые данные.

---

## \*\*1. Устройство VAE\*\*

VAE состоит из трёх ключевых компонентов:

### \*\*1. Энкодер (Encoder)\*\*

- Преобразует входные данные \*\*x\*\* в параметры распределения в латентном пространстве:

- \*\*μ (mean)\*\* — вектор средних значений.

- \*\*σ (log variance)\*\* — логарифм дисперсии (для устойчивости).

- Архитектура: полносвязная или свёрточная сеть.

### \*\*2. Сэмплирование (Reparameterization Trick)\*\*

- Генерирует латентный вектор \*\*z\*\* по формуле:

\[

z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)

\]

где:

- \*\*ϵ\*\* — шум из стандартного нормального распределения.

- \*\*⊙\*\* — поэлементное умножение.

### \*\*3. Декодер (Decoder)\*\*

- Восстанавливает данные \*\*x̂\*\* из латентного вектора \*\*z\*\*.

- Архитектура: зеркальна энкодеру.

---

## \*\*2. Функция потерь VAE\*\*

Функция потерь VAE состоит из двух частей:

### \*\*1. Reconstruction Loss (MSE или BCE)\*\*

- Ошибка восстановления данных:

\[

\mathcal{L}\_{\text{recon}} = \mathbb{E}\_{q(z|x)}[\log p(x|z)]

\]

- Для изображений: \*\*бинарная кросс-энтропия (BCE)\*\*.

- Для непрерывных данных: \*\*MSE\*\*.

### \*\*2. KL-дивергенция (Kullback-Leibler Divergence)\*\*

- "Штраф" за отличие латентного распределения от стандартного нормального:

\[

\mathcal{L}\_{\text{KL}} = D\_{\text{KL}}(q(z|x) \, \| \, p(z))

\]

где:

- \*\*q(z|x)\*\* — распределение энкодера.

- \*\*p(z) = 𝒩(0, I)\*\* — стандартное нормальное распределение.

### \*\*Итоговая функция потерь\*\*

\[

\mathcal{L} = \mathcal{L}\_{\text{recon}}} + \beta \cdot \mathcal{L}\_{\text{KL}}

\]

- \*\*β\*\* — гиперпараметр, балансирующий влияние KL-дивергенции.

---

## \*\*3. Ошибка Кульбака-Лейблера (KL Divergence)\*\*

### \*\*Формула для VAE\*\*

Для гауссова распределения \*\*q(z|x) = 𝒩(μ, σ²)\*\* и \*\*p(z) = 𝒩(0, 1)\*\*:

\[

D\_{\text{KL}} = -\frac{1}{2} \sum\_{i=1}^{d} \left(1 + \log(\sigma\_i^2) - \mu\_i^2 - \sigma\_i^2 \right)

\]

где \*\*d\*\* — размерность латентного пространства.

### \*\*Смысл KL-дивергенции\*\*

- \*\*μ → 0, σ → 1\*\*: Латенты стремятся к стандартному нормальному распределению.

- \*\*σ → 0\*\*: Провал коллапса (latent collapse) — модель игнорирует латентное пространство.

---

## \*\*4. Пример реализации VAE на PyTorch\*\*

```python

import torch

import torch.nn as nn

class VAE(nn.Module):

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

super().\_\_init\_\_()

# Энкодер

self.encoder = nn.Sequential(

nn.Linear(input\_dim, 128),

nn.ReLU(),

nn.Linear(128, 64),

nn.ReLU(),

)

self.fc\_mu = nn.Linear(64, latent\_dim)

self.fc\_logvar = nn.Linear(64, latent\_dim)

# Декодер

self.decoder = nn.Sequential(

nn.Linear(latent\_dim, 64),

nn.ReLU(),

nn.Linear(64, 128),

nn.ReLU(),

nn.Linear(128, input\_dim),

nn.Sigmoid(),

)

def encode(self, x):

h = self.encoder(x)

return self.fc\_mu(h), self.fc\_logvar(h)

def reparameterize(self, mu, logvar):

std = torch.exp(0.5 \* logvar)

eps = torch.randn\_like(std)

return mu + eps \* std

def forward(self, x):

mu, logvar = self.encode(x)

z = self.reparameterize(mu, logvar)

x\_recon = self.decoder(z)

return x\_recon, mu, logvar

# Функция потерь

def vae\_loss(x, x\_recon, mu, logvar):

BCE = nn.functional.binary\_cross\_entropy(x\_recon, x, reduction='sum')

KLD = -0.5 \* torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

return BCE + KLD

```

---

## \*\*5. Применение VAE\*\*

- \*\*Генерация данных\*\*: Создание новых изображений, текстов, музыки.

- \*\*Денойзинг\*\*: Удаление шума из данных.

- \*\*Сжатие данных\*\*: Латентное пространство как компактное представление.

---

## \*\*6. Проблемы VAE\*\*

- \*\*Размытость генераций\*\*: Из-за MSE/BCE выходы часто "усреднённые".

- \*\*Latent Collapse\*\*: KL-дивергенция может подавить латентное пространство.

- \*\*Баланс β\*\*: Слишком большой β ухудшает качество реконструкции.

---

## \*\*7. Вывод\*\*

- \*\*VAE\*\* — это генеративная модель с вероятностным латентным пространством.

- \*\*KL-дивергенция\*\* регулирует близость латентных переменных к 𝒩(0, I).

- \*\*Функция потерь\*\*: Reconstruction Loss + β·KL.

- \*\*Пример использования\*\*: Генерация изображений, сжатие данных.

\*\*Пример ответа на экзамене:\*\*

> \*"VAE состоит из энкодера (предсказывает μ и σ), сэмплирования (reparameterization trick) и декодера. Функция потерь включает ошибку восстановления и KL-дивергенцию, которая штрафует отклонение латентного распределения от 𝒩(0, I). KL-дивергенция вычисляется как: \(-\frac{1}{2} \sum (1 + \log(\sigma^2) - \mu^2 - \sigma^2)\). VAE применяется для генерации и сжатия данных."\*

Вопрос № 37 Инструмент Callback’ов. Встроенные Callback’и keras’а.

### \*\*Инструмент Callback’ов в Keras. Встроенные Callback’и\*\*

\*\*Callback’и\*\* — это функции или объекты, которые вызываются во время обучения модели (на старте, в конце эпохи, после батча и т.д.). Они позволяют:

- \*\*Контролировать процесс обучения\*\* (остановка при переобучении, изменение LR).

- \*\*Сохранять модель\*\* на лучших эпохах.

- \*\*Логировать метрики\*\* (TensorBoard, CSV).

---

## \*\*1. Основные Callback’и в Keras\*\*

Keras предоставляет набор встроенных callback’ов, которые можно передать в `model.fit()`:

### \*\*1. `ModelCheckpoint` — сохранение модели\*\*

Сохраняет модель (или веса) в указанную директорию.

\*\*Параметры:\*\*

- \*\*`filepath`\*\* — путь для сохранения.

- \*\*`monitor`\*\* — метрика для отслеживания (например, `val\_loss`).

- \*\*`save\_best\_only`\*\* — сохранять только лучшую модель.

- \*\*`mode`\*\* — `min` (для loss) или `max` (для accuracy).

```python

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint(

filepath='best\_model.keras',

monitor='val\_loss',

save\_best\_only=True,

mode='min'

)

```

### \*\*2. `EarlyStopping` — ранняя остановка\*\*

Останавливает обучение, если метрика не улучшается.

\*\*Параметры:\*\*

- \*\*`monitor`\*\* — отслеживаемая метрика.

- \*\*`patience`\*\* — число эпох без улучшения перед остановкой.

- \*\*`restore\_best\_weights`\*\* — возврат к лучшим весам.

```python

from keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5,

restore\_best\_weights=True

)

```

### \*\*3. `ReduceLROnPlateau` — динамическое изменение LR\*\*

Уменьшает learning rate, если метрика не улучшается.

\*\*Параметры:\*\*

- \*\*`monitor`\*\* — метрика.

- \*\*`factor`\*\* — множитель уменьшения (например, `0.1`).

- \*\*`patience`\*\* — число эпох перед уменьшением.

```python

from keras.callbacks import ReduceLROnPlateau

reduce\_lr = ReduceLROnPlateau(

monitor='val\_loss',

factor=0.1,

patience=3

)

```

### \*\*4. `TensorBoard` — визуализация обучения\*\*

Логирует метрики для TensorBoard.

```python

from keras.callbacks import TensorBoard

tensorboard = TensorBoard(

log\_dir='./logs',

histogram\_freq=1 # Частота записи гистограмм весов

)

```

### \*\*5. `CSVLogger` — сохранение метрик в CSV\*\*

```python

from keras.callbacks import CSVLogger

csv\_logger = CSVLogger('training\_log.csv')

```

### \*\*6. `LearningRateScheduler` — ручное изменение LR\*\*

Позволяет задать свой график изменения learning rate.

```python

from keras.callbacks import LearningRateScheduler

def lr\_schedule(epoch):

return 0.001 \* (0.9 \*\* epoch)

lr\_scheduler = LearningRateScheduler(lr\_schedule)

```

---

## \*\*2. Как передать Callback’и в `model.fit()`\*\*

```python

callbacks\_list = [

ModelCheckpoint('best\_model.keras'),

EarlyStopping(patience=5),

TensorBoard('./logs'),

CSVLogger('training\_log.csv')

]

model.fit(

X\_train, y\_train,

validation\_data=(X\_val, y\_val),

epochs=50,

callbacks=callbacks\_list

)

```

---

## \*\*3. Создание своего Callback’а\*\*

Чтобы создать кастомный callback, нужно унаследоваться от `keras.callbacks.Callback` и переопределить методы:

- \*\*`on\_epoch\_begin`\*\* / \*\*`on\_epoch\_end`\*\*

- \*\*`on\_batch\_begin`\*\* / \*\*`on\_batch\_end`\*\*

- \*\*`on\_train\_begin`\*\* / \*\*`on\_train\_end`\*\*

\*\*Пример:\*\*

```python

from keras.callbacks import Callback

class CustomCallback(Callback):

def on\_epoch\_end(self, epoch, logs=None):

if logs['val\_accuracy'] > 0.95:

print("Достигнута точность 95%, остановка обучения!")

self.model.stop\_training = True

custom\_cb = CustomCallback()

```

---

## \*\*4. Вывод\*\*

✅ \*\*Встроенные callback’и Keras:\*\*

- `ModelCheckpoint` — сохранение модели.

- `EarlyStopping` — остановка при переобучении.

- `ReduceLROnPlateau` — адаптивный learning rate.

- `TensorBoard` / `CSVLogger` — логирование метрик.

✅ \*\*Кастомные callback’и:\*\*

- Наследование от `Callback` и переопределение методов.

\*\*Пример ответа на экзамене:\*\*

> \*"Callback’и в Keras позволяют контролировать обучение модели. Основные встроенные callback’и: `ModelCheckpoint` (сохранение модели), `EarlyStopping` (ранняя остановка), `ReduceLROnPlateau` (адаптивный LR) и `TensorBoard` (визуализация). Можно создавать свои callback’и, наследуясь от `keras.callbacks.Callback`."\*

Вопрос № 38 Архитектура U-Net. Принцип построения.

### \*\*Архитектура U-Net: принцип построения и особенности\*\*

\*\*U-Net\*\* — это свёрточная нейронная сеть, разработанная для \*\*сегментации изображений\*\* (например, в медицинских данных). Её ключевая особенность — \*\*U-образная структура\*\* с кодировщиком (энкодером) и декодером, соединёнными \*\*skip-connections\*\*.

---

## \*\*1. Структура U-Net\*\*

Архитектура состоит из двух основных частей:

### \*\*1. Энкодер (Contracting Path — нисходящий путь)\*\*

- \*\*Цель\*\*: Уменьшение пространственного разрешения и выделение признаков.

- \*\*Состоит из блоков\*\*:

- \*\*Свёртка (Conv2D) + ReLU\*\*

- \*\*MaxPooling\*\* (уменьшение размера в 2 раза).

- Пример блока:

```python

x = Conv2D(64, (3, 3), activation='relu', padding='same')(input)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

pooled = MaxPooling2D((2, 2))(x)

```

### \*\*2. Декодер (Expansive Path — восходящий путь)\*\*

- \*\*Цель\*\*: Увеличение разрешения и точное позиционирование объектов.

- \*\*Состоит из блоков\*\*:

- \*\*Upsampling\*\* (или \*\*Transposed Convolution\*\*).

- \*\*Конкатенация\*\* с соответствующим слоем энкодера (skip-connection).

- \*\*Свёртка + ReLU\*\*.

- Пример блока:

```python

x = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(input)

x = concatenate([x, skip\_connection], axis=-1) # Объединение с skip-connection

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)

```

### \*\*3. Skip-connections\*\*

- \*\*Соединяют\*\* слои энкодера и декодера \*\*на одинаковых уровнях\*\*.

- \*\*Помогают\*\* сохранить \*\*пространственные детали\*\*, потерянные при пулинге.

---

## \*\*2. Принцип построения U-Net\*\*

1. \*\*Энкодер\*\*:

- Каждый блок уменьшает размерность (например, с `256x256` до `128x128`).

- Увеличивает количество фильтров (например, `64 → 128 → 256`).

2. \*\*Декодер\*\*:

- Каждый блок увеличивает размерность (например, с `16x16` до `32x32`).

- Уменьшает количество фильтров (например, `512 → 256 → 128`).

3. \*\*Skip-connections\*\*:

- Передают \*\*высокоуровневые признаки\*\* из энкодера в декодер.

4. \*\*Выходной слой\*\*:

- Обычно `Conv2D(1, (1, 1), activation='sigmoid')` для бинарной сегментации.

---

## \*\*3. Пример реализации на Keras\*\*

```python

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate

from tensorflow.keras.models import Model

def unet(input\_size=(256, 256, 1)):

inputs = Input(input\_size)

# Энкодер

c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)

c1 = Conv2D(64, (3, 3), activation='relu', padding='same')(c1)

p1 = MaxPooling2D((2, 2))(c1)

c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(p1)

c2 = Conv2D(128, (3, 3), activation='relu', padding='same')(c2)

p2 = MaxPooling2D((2, 2))(c2)

# Центральный блок

c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(p2)

c3 = Conv2D(256, (3, 3), activation='relu', padding='same')(c3)

# Декодер

u4 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c3)

u4 = concatenate([u4, c2])

c4 = Conv2D(128, (3, 3), activation='relu', padding='same')(u4)

c4 = Conv2D(128, (3, 3), activation='relu', padding='same')(c4)

u5 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c4)

u5 = concatenate([u5, c1])

c5 = Conv2D(64, (3, 3), activation='relu', padding='same')(u5)

c5 = Conv2D(64, (3, 3), activation='relu', padding='same')(c5)

# Выход

outputs = Conv2D(1, (1, 1), activation='sigmoid')(c5)

return Model(inputs=inputs, outputs=outputs)

model = unet()

model.compile(optimizer='adam', loss='binary\_crossentropy')

```

---

## \*\*4. Применение U-Net\*\*

- \*\*Медицинская сегментация\*\* (опухоли, органы на МРТ).

- \*\*Детекция объектов\*\* (спутниковые снимки).

- \*\*Генерация масок\*\* (для автономного вождения).

---

## \*\*5. Плюсы и минусы\*\*

✅ \*\*Преимущества\*\*:

- Высокая точность даже на небольших датасетах.

- Сохранение пространственных деталей благодаря skip-connections.

❌ \*\*Недостатки\*\*:

- Большое число параметров (риск переобучения).

- Требует значительных вычислительных ресурсов.

---

## \*\*6. Вывод\*\*

- \*\*U-Net\*\* — это симметричная архитектура с энкодером, декодером и skip-connections.

- \*\*Энкодер\*\* уменьшает разрешение, \*\*декодер\*\* — восстанавливает.

- \*\*Skip-connections\*\* передают детали из энкодера в декодер.

- \*\*Применение\*\*: сегментация изображений, особенно в медицине.

\*\*Пример ответа на экзамене:\*\*

> \*"U-Net состоит из энкодера (свёртки + пулинг), декодера (транспонированные свёртки + конкатенация с skip-connections) и выходного слоя. Skip-connections помогают сохранить детали изображения. Применяется для задач сегментации, особенно в медицине."\*

Вопрос № 39 Задача сегментации изображений. Формирование выборки.

### \*\*Задача сегментации изображений и формирование выборки\*\*

\*\*Сегментация изображений\*\* — это задача разделения изображения на семантически значимые части (например, объекты, классы пикселей). Она широко применяется в медицине, автономном вождении, спутниковом анализе и др.

---

## \*\*1. Типы сегментации\*\*

1. \*\*Бинарная сегментация\*\* (1 класс + фон).

- Пример: выделение опухоли на МРТ.

2. \*\*Мультиклассовая сегментация\*\* (несколько классов).

- Пример: разметка дорог, зданий, деревьев на карте.

3. \*\*Инстанс-сегментация\*\* (разделение объектов одного класса).

- Пример: выделение каждого человека в толпе.

---

## \*\*2. Формирование выборки для сегментации\*\*

### \*\*1. Данные\*\*

- \*\*Изображения (X)\*\*: Оригинальные снимки (например, медицинские, спутниковые).

- \*\*Маски (Y)\*\*: Размеченные изображения, где каждому пикселю присвоен класс.

\*\*Пример маски\*\*:

- Бинарная: `0` — фон, `1` — объект.

- Мультиклассовая: `0` — фон, `1` — класс 1, `2` — класс 2.

### \*\*2. Подготовка данных\*\*

#### \*\*1. Сбор и разметка\*\*

- \*\*Инструменты для разметки\*\*:

- LabelMe, CVAT, Supervisely.

- Для медицинских данных: ITK-SNAP, 3D Slicer.

- \*\*Требования к маскам\*\*:

- Точное соответствие пикселей исходному изображению.

- Форматы: `.png`, `.npy` (для медицинских данных).

#### \*\*2. Предобработка\*\*

- \*\*Нормализация изображений\*\*:

```python

image = image / 255.0 # Приведение к [0, 1]

```

- \*\*Аугментация\*\* (для увеличения выборки):

- Повороты, сдвиги, отражения.

- Для масок аугментация должна быть \*\*согласована\*\* с изображениями!

\*\*Пример аугментации в Keras\*\*:

```python

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=20,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True

)

# Для изображений и масок используются одинаковые параметры!

train\_generator = datagen.flow(x\_train, y\_train, batch\_size=32)

```

#### \*\*3. Разделение на train/val/test\*\*

- \*\*Обычное соотношение\*\*: 70%/15%/15%.

- \*\*Важно\*\*: Разделять \*\*по сценам/пациентам\*\*, а не случайно (для избежания утечки данных).

---

## \*\*3. Форматы данных\*\*

### \*\*1. Для классических задач\*\*

- \*\*Изображения\*\*: `.jpg`, `.png`.

- \*\*Маски\*\*: `.png` (1 канал с индексами классов).

### \*\*2. Для медицинских данных\*\*

- \*\*Изображения\*\*: DICOM, NIfTI.

- \*\*Маски\*\*: `.npy` или `.nrrd`.

### \*\*3. Подготовка тензоров\*\*

- \*\*Изображения\*\*: `(batch\_size, height, width, channels)`.

- \*\*Маски\*\*: `(batch\_size, height, width, 1)` (для бинарной) или `(batch\_size, height, width, num\_classes)` (one-hot).

\*\*Конвертация в one-hot\*\*:

```python

from keras.utils import to\_categorical

y\_train\_onehot = to\_categorical(y\_train, num\_classes=3) # 3 класса

```

---

## \*\*4. Архитектуры моделей для сегментации\*\*

1. \*\*U-Net\*\*:

- Энкодер + декодер + skip-connections.

- Подходит для небольших датасетов.

2. \*\*DeepLab\*\*:

- Использует атрофированные свёртки (Atrous Convolutions).

3. \*\*Mask R-CNN\*\*:

- Для инстанс-сегментации.

\*\*Пример загрузки данных для U-Net\*\*:

```python

def load\_data(image\_paths, mask\_paths):

images = []

masks = []

for img\_path, mask\_path in zip(image\_paths, mask\_paths):

img = cv2.imread(img\_path, cv2.IMREAD\_COLOR)

mask = cv2.imread(mask\_path, cv2.IMREAD\_GRAYSCALE)

images.append(img)

masks.append(mask)

return np.array(images), np.array(masks)

X\_train, y\_train = load\_data(train\_images, train\_masks)

```

---

## \*\*5. Метрики качества\*\*

- \*\*IoU (Intersection over Union)\*\*:

\[

\text{IoU} = \frac{\text{Пересечение}}{\text{Объединение}}

\]

- \*\*Dice Coefficient\*\*:

\[

\text{Dice} = \frac{2 \cdot \text{Пересечение}}{\text{Общая площадь}}

\]

- \*\*Pixel Accuracy\*\*: Доля правильно классифицированных пикселей.

\*\*Пример вычисления IoU в Keras\*\*:

```python

def iou(y\_true, y\_pred):

y\_true = tf.cast(y\_true, tf.float32)

y\_pred = tf.cast(y\_pred > 0.5, tf.float32)

intersection = tf.reduce\_sum(y\_true \* y\_pred)

union = tf.reduce\_sum(y\_true) + tf.reduce\_sum(y\_pred) - intersection

return (intersection + 1e-7) / (union + 1e-7)

```

---

## \*\*6. Вывод\*\*

1. \*\*Сегментация\*\* требует пар: \*\*изображение + маска\*\*.

2. \*\*Предобработка\*\*: нормализация, аугментация, one-hot кодирование.

3. \*\*Разделение данных\*\*: по сценам, а не случайное.

4. \*\*Метрики\*\*: IoU, Dice, Pixel Accuracy.

\*\*Пример ответа на экзамене\*\*:

> \*"Для задачи сегментации формируется выборка из изображений и масок (размеченных пикселей). Данные предобрабатываются: нормализуются, аугментируются и разделяются на train/val/test. Маски могут быть бинарными или one-hot. Популярные архитектуры — U-Net и DeepLab. Метрики качества: IoU и Dice Coefficient."\*

Вопрос № 40 Задача кластеризации. Алгоритм Kmean.

### \*\*Задача кластеризации и алгоритм K-means\*\*

\*\*Кластеризация\*\* — это задача машинного обучения без учителя, цель которой — разбить данные на группы (\*\*кластеры\*\*) так, чтобы объекты внутри одного кластера были \*\*похожи\*\*, а между разными кластерами — \*\*отличались\*\*.

\*\*K-means\*\* — один из самых популярных алгоритмов кластеризации, основанный на минимизации внутрикластерного расстояния до центроидов.

---

## \*\*1. Принцип работы K-means\*\*

### \*\*1. Инициализация\*\*

- Выбирается число кластеров \*\*K\*\*.

- Случайно или специальным способом (K-means++) задаются начальные \*\*центроиды\*\* (центры кластеров).

### \*\*2. Основные шаги алгоритма\*\*

1. \*\*Назначение кластеров\*\*:

- Каждый объект присваивается к ближайшему центроиду (по расстоянию, обычно Евклидову).

2. \*\*Пересчёт центроидов\*\*:

- Новый центроид вычисляется как среднее всех точек кластера.

3. \*\*Повторение\*\*:

- Шаги 1 и 2 повторяются, пока центроиды не перестанут значительно меняться.

### \*\*3. Формальное описание\*\*

Алгоритм минимизирует \*\*сумму квадратов расстояний\*\* (within-cluster sum of squares, WCSS):

\[

\text{WCSS} = \sum\_{i=1}^{K} \sum\_{x \in C\_i} \|x - \mu\_i\|^2

\]

где:

- \(K\) — число кластеров,

- \(C\_i\) — точки кластера \(i\),

- \(\mu\_i\) — центроид кластера \(i\).

---

## \*\*2. Реализация K-means в Python\*\*

### \*\*1. С использованием `sklearn`\*\*

```python

from sklearn.cluster import KMeans

import numpy as np

# Пример данных

X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])

# Создание модели

kmeans = KMeans(n\_clusters=2, random\_state=42)

kmeans.fit(X)

# Результаты

print("Центроиды:", kmeans.cluster\_centers\_)

print("Метки кластеров:", kmeans.labels\_)

```

### \*\*2. Визуализация\*\*

```python

import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels\_, cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=200, c='red', marker='X')

plt.show()

```

---

## \*\*3. Выбор числа кластеров (K)\*\*

### \*\*1. Метод локтя (Elbow Method)\*\*

- Строится график зависимости WCSS от числа кластеров.

- Оптимальное \(K\) — точка изгиба ("локоть").

\*\*Пример:\*\*

```python

wcss = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss, marker='o')

plt.xlabel('Число кластеров (K)')

plt.ylabel('WCSS')

plt.show()

```

### \*\*2. Silhouette Score\*\*

- Оценивает, насколько хорошо объекты кластеризованы.

- Диапазон: от \(-1\) (плохо) до \(1\) (хорошо).

\*\*Пример:\*\*

```python

from sklearn.metrics import silhouette\_score

silhouette\_scores = []

for k in range(2, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

labels = kmeans.fit\_predict(X)

score = silhouette\_score(X, labels)

silhouette\_scores.append(score)

plt.plot(range(2, 11), silhouette\_scores, marker='o')

plt.xlabel('Число кластеров (K)')

plt.ylabel('Silhouette Score')

plt.show()

```

---

## \*\*4. Плюсы и минусы K-means\*\*

✅ \*\*Преимущества\*\*:

- Простота реализации и интерпретации.

- Быстрая работа на больших данных.

- Масштабируемость.

❌ \*\*Недостатки\*\*:

- Требует задания числа кластеров \(K\).

- Чувствителен к начальным центроидам (может сходиться к локальным минимумам).

- Плохо работает с кластерами сложной формы.

---

## \*\*5. Области применения\*\*

- \*\*Сегментация клиентов\*\* (CRM-аналитика).

- \*\*Анализ изображений\*\* (группировка пикселей по цвету).

- \*\*Биоинформатика\*\* (кластеризация генов).

---

## \*\*6. Вывод\*\*

- \*\*K-means\*\* — алгоритм кластеризации, основанный на центроидах.

- \*\*Шаги\*\*: инициализация, назначение кластеров, пересчёт центроидов.

- \*\*Выбор \(K\)\*\*: метод локтя, silhouette score.

- \*\*Применение\*\*: маркетинг, компьютерное зрение, биоинформатика.

\*\*Пример ответа на экзамене:\*\*

> \*"K-means — это алгоритм кластеризации, который разбивает данные на K кластеров, минимизируя сумму квадратов расстояний до центроидов. Основные шаги: инициализация центроидов, назначение точек кластерам и пересчёт центроидов. Для выбора K используют метод локтя или silhouette score. Применяется в сегментации клиентов и анализе изображений."\*

Вопрос № 41 Рекуррентные нейронные сети. Слой LSTM.

### \*\*Рекуррентные нейронные сети (RNN) и слой LSTM\*\*

\*\*Рекуррентные нейронные сети (RNN)\*\* — это класс нейронных сетей, предназначенных для обработки \*\*последовательностей данных\*\* (текст, временные ряды, речь). Они учитывают \*\*контекст\*\* предыдущих элементов последовательности.

\*\*LSTM (Long Short-Term Memory)\*\* — усовершенствованный тип RNN, решающий проблему \*\*исчезающего градиента\*\* и способный запоминать долгосрочные зависимости.

---

## \*\*1. Устройство RNN\*\*

### \*\*Основная идея\*\*

- На каждом шаге RNN принимает:

- \*\*Текущий вход\*\* \(x\_t\) (например, слово или значение временного ряда).

- \*\*Скрытое состояние\*\* \(h\_{t-1}\) (информация о предыдущих шагах).

- Возвращает:

- \*\*Выход\*\* \(y\_t\) (предсказание).

- \*\*Новое скрытое состояние\*\* \(h\_t\).

### \*\*Проблема стандартных RNN\*\*

- \*\*Исчезающий градиент\*\*: При обучении на длинных последовательностях градиенты могут становиться очень маленькими, что мешает обучению.

- \*\*Краткосрочная память\*\*: Сложно запоминать контекст, который был много шагов назад.

---

## \*\*2. Слой LSTM\*\*

LSTM решает проблемы RNN за счёт \*\*ячеек памяти\*\* и \*\*трёх "ворот"\*\*, регулирующих поток информации:

### \*\*1. Структура LSTM\*\*

- \*\*Ячейка памяти (cell state, \(C\_t\))\*\* — хранит долгосрочную информацию.

- \*\*Скрытое состояние (hidden state, \(h\_t\))\*\* — передаёт краткосрочную информацию.

- \*\*Три управляющих "ворота"\*\*:

1. \*\*Забывающий гейт (Forget Gate, \(f\_t\))\*\* — решает, какую информацию удалить из \(C\_{t-1}\).

2. \*\*Входной гейт (Input Gate, \(i\_t\))\*\* — выбирает, какую новую информацию записать в \(C\_t\).

3. \*\*Выходной гейт (Output Gate, \(o\_t\))\*\* — определяет, какую информацию передать в \(h\_t\).

### \*\*2. Формулы LSTM\*\*

\[

\begin{aligned}

f\_t &= \sigma(W\_f \cdot [h\_{t-1}, x\_t] + b\_f) \\

i\_t &= \sigma(W\_i \cdot [h\_{t-1}, x\_t] + b\_i) \\

\tilde{C}\_t &= \tanh(W\_C \cdot [h\_{t-1}, x\_t] + b\_C) \\

C\_t &= f\_t \odot C\_{t-1} + i\_t \odot \tilde{C}\_t \\

o\_t &= \sigma(W\_o \cdot [h\_{t-1}, x\_t] + b\_o) \\

h\_t &= o\_t \odot \tanh(C\_t)

\end{aligned}

\]

Где:

- \(\sigma\) — сигмоида (активация гейтов),

- \(\odot\) — поэлементное умножение,

- \(\tanh\) — гиперболический тангенс (для значений ячейки).

---

## \*\*3. Реализация LSTM в Keras\*\*

### \*\*1. Простейшая модель для предсказания временных рядов\*\*

```python

from keras.models import Sequential

from keras.layers import LSTM, Dense

model = Sequential([

LSTM(64, input\_shape=(10, 1)), # 10 временных шагов, 1 признак

Dense(1) # Выход — одно число

])

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

```

### \*\*2. Многослойная LSTM\*\*

```python

model = Sequential([

LSTM(64, return\_sequences=True, input\_shape=(10, 1)), # Возвращает последовательность

LSTM(32), # Второй слой

Dense(1)

])

```

### \*\*3. Обработка текста (NLP)\*\*

```python

model = Sequential([

Embedding(input\_dim=10000, output\_dim=128), # Векторизация слов

LSTM(64),

Dense(1, activation='sigmoid') # Классификация

])

```

---

## \*\*4. Параметры слоя LSTM в Keras\*\*

| Параметр | Описание |

|------------------------|----------|

| \*\*`units`\*\* | Количество нейронов. |

| \*\*`return\_sequences`\*\* | Если `True`, возвращает все скрытые состояния (для стека LSTM). |

| \*\*`return\_state`\*\* | Возвращает последнее состояние \(h\_t\) и \(C\_t\). |

| \*\*`dropout`\*\* | Дропаут на входе. |

| \*\*`recurrent\_dropout`\*\*| Дропаут на рекуррентных связях. |

---

## \*\*5. Плюсы и минусы LSTM\*\*

✅ \*\*Преимущества\*\*:

- Запоминает долгосрочные зависимости.

- Устойчив к проблеме исчезающего градиента.

- Подходит для текста, временных рядов, речи.

❌ \*\*Недостатки\*\*:

- Вычислительно сложнее, чем RNN.

- Требует больше данных для обучения.

---

## \*\*6. Области применения\*\*

- \*\*Прогнозирование временных рядов\*\* (биржа, погода).

- \*\*Обработка текста\*\* (машинный перевод, генерация).

- \*\*Распознавание речи\*\* (ASR-системы).

---

## \*\*7. Вывод\*\*

- \*\*LSTM\*\* — это RNN с ячейками памяти и управляемыми воротами.

- \*\*Гейты\*\*: Forget, Input, Output (контролируют информацию).

- \*\*Реализация\*\*: В Keras через `LSTM(units, return\_sequences)`.

- \*\*Применение\*\*: NLP, временные ряды, речь.

\*\*Пример ответа на экзамене:\*\*

> \*"LSTM — это тип RNN с ячейкой памяти и тремя гейтами (Forget, Input, Output), которые решают проблему долгосрочных зависимостей. Формулы включают сигмоиду для гейтов и tanh для значений ячейки. В Keras используется слой `LSTM(units)`, где `units` — число нейронов. Применяется в NLP, прогнозировании и анализе временных рядов."\*

Вопрос № 42 Устройство GAN.

### \*\*Устройство Generative Adversarial Networks (GAN)\*\*

\*\*Generative Adversarial Networks (GAN)\*\* — это архитектура глубокого обучения, состоящая из двух конкурирующих нейронных сетей: \*\*генератора\*\* и \*\*дискриминатора\*\*, которые обучаются вместе в процессе "игры".

---

## \*\*1. Основные компоненты GAN\*\*

### \*\*1. Генератор (Generator)\*\*

- \*\*Цель\*\*: Создавать реалистичные данные (изображения, текст, аудио) из случайного шума.

- \*\*Вход\*\*: Вектор шума \( z \) (обычно из нормального распределения).

- \*\*Выход\*\*: Сгенерированные данные (например, изображение 64x64x3).

- \*\*Архитектура\*\*: Обычно использует \*\*транспонированные свёртки\*\* (для изображений).

### \*\*2. Дискриминатор (Discriminator)\*\*

- \*\*Цель\*\*: Отличать реальные данные от сгенерированных.

- \*\*Вход\*\*: Реальные данные \( x \) или сгенерированные \( G(z) \).

- \*\*Выход\*\*: Вероятность того, что данные настоящие (от 0 до 1).

- \*\*Архитектура\*\*: Свёрточная сеть (для изображений).

---

## \*\*2. Принцип работы GAN\*\*

1. \*\*Генератор\*\* создаёт фейковые данные \( G(z) \).

2. \*\*Дискриминатор\*\* получает:

- Реальные данные \( x \) (с меткой 1).

- Фейковые данные \( G(z) \) (с меткой 0).

3. \*\*Дискриминатор обучается\*\* максимизировать точность классификации:

\[

\max\_D \mathbb{E}\_{x \sim p\_{\text{data}}}[\log D(x)] + \mathbb{E}\_{z \sim p\_z}[\log (1 - D(G(z)))]

\]

4. \*\*Генератор обучается\*\* обманывать дискриминатор:

\[

\min\_G \mathbb{E}\_{z \sim p\_z}[\log (1 - D(G(z)))]

\]

5. Процесс повторяется до тех пор, пока генератор не начнёт создавать правдоподобные данные.

---

## \*\*3. Архитектуры GAN\*\*

### \*\*1. DCGAN (Deep Convolutional GAN)\*\*

- Использует \*\*свёрточные слои\*\* в генераторе и дискриминаторе.

- \*\*Ключевые особенности\*\*:

- Замена пулинга на \*\*страйдинг\*\* (strided convolutions).

- \*\*Batch Normalization\*\* в обоих сетях.

- \*\*LeakyReLU\*\* в дискриминаторе.

### \*\*2. Conditional GAN (cGAN)\*\*

- Добавляет \*\*условную информацию\*\* (например, класс изображения).

- Пример: Генерация определённой цифры в MNIST.

### \*\*3. CycleGAN\*\*

- Преобразует изображения из одного домена в другой (например, лошадь → зебра).

---

## \*\*4. Реализация GAN на Python (Keras)\*\*

### \*\*1. Генератор\*\*

```python

from keras.models import Sequential

from keras.layers import Dense, Reshape, Conv2DTranspose

generator = Sequential([

Dense(128 \* 8 \* 8, input\_dim=100), # Вход: шум (100-мерный вектор)

Reshape((8, 8, 128)),

Conv2DTranspose(64, (4, 4), strides=2, padding='same', activation='relu'),

Conv2DTranspose(3, (4, 4), strides=2, padding='same', activation='tanh') # Выход: изображение 32x32x3

])

```

### \*\*2. Дискриминатор\*\*

```python

from keras.layers import Conv2D, LeakyReLU, Flatten

discriminator = Sequential([

Conv2D(64, (4, 4), strides=2, padding='same', input\_shape=(32, 32, 3)),

LeakyReLU(0.2),

Conv2D(128, (4, 4), strides=2, padding='same'),

LeakyReLU(0.2),

Flatten(),

Dense(1, activation='sigmoid') # Выход: вероятность "реальности"

])

```

### \*\*3. Объединённая модель GAN\*\*

```python

from keras.models import Model

from keras.optimizers import Adam

discriminator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

discriminator.trainable = False # Замораживаем дискриминатор при обучении генератора

gan\_input = Input(shape=(100,))

fake\_image = generator(gan\_input)

gan\_output = discriminator(fake\_image)

gan = Model(gan\_input, gan\_output)

gan.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

```

---

## \*\*5. Обучение GAN\*\*

### \*\*1. Алгоритм обучения\*\*

1. \*\*Выбираем батч реальных изображений\*\*.

2. \*\*Генерируем батч фейковых изображений\*\*.

3. \*\*Обучаем дискриминатор\*\* на реальных и фейковых данных.

4. \*\*Обучаем генератор\*\* (через объединённую модель GAN).

### \*\*2. Код обучения\*\*

```python

for epoch in range(epochs):

for batch in range(num\_batches):

# 1. Обучаем дискриминатор

real\_images = get\_real\_images\_batch()

noise = np.random.normal(0, 1, (batch\_size, 100))

fake\_images = generator.predict(noise)

d\_loss\_real = discriminator.train\_on\_batch(real\_images, np.ones(batch\_size))

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, np.zeros(batch\_size))

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# 2. Обучаем генератор

noise = np.random.normal(0, 1, (batch\_size, 100))

g\_loss = gan.train\_on\_batch(noise, np.ones(batch\_size)) # Обманываем дискриминатор

```

---

## \*\*6. Проблемы GAN\*\*

### \*\*1. Несходимость\*\*

- Дискриминатор или генератор могут "подавить" друг друга.

### \*\*2. Mode Collapse\*\*

- Генератор начинает производить \*\*одно и то же\*\* изображение.

### \*\*3. Оценка качества\*\*

- Нет чётких метрик (часто используют \*\*Inception Score\*\* или \*\*FID\*\*).

---

## \*\*7. Применение GAN\*\*

- \*\*Генерация изображений\*\* (StyleGAN, BigGAN).

- \*\*Фотореалистичные аватары\*\* (DeepFake).

- \*\*Улучшение изображений\*\* (Super-Resolution).

---

## \*\*8. Вывод\*\*

- \*\*GAN\*\* состоит из \*\*генератора\*\* и \*\*дискриминатора\*\*, соревнующихся друг с другом.

- \*\*Генератор\*\* учится создавать реалистичные данные, \*\*дискриминатор\*\* — отличать фейки от реальных.

- \*\*Популярные архитектуры\*\*: DCGAN, cGAN, CycleGAN.

- \*\*Проблемы\*\*: Mode Collapse, несходимость.

\*\*Пример ответа на экзамене:\*\*

> \*"GAN — это архитектура из генератора и дискриминатора, которые обучаются вместе. Генератор создаёт данные из шума, а дискриминатор пытается отличить их от реальных. Оптимизация идёт через минимаксную игру: генератор минимизирует log(1 - D(G(z))), а дискриминатор максимизирует точность классификации. Применяется для генерации изображений, стилизации и улучшения данных."\*

Вопрос № 43 GAN с условием.

### \*\*Условные генеративно-состязательные сети (Conditional GAN, cGAN)\*\*

\*\*Conditional GAN (cGAN)\*\* — это разновидность GAN, в которой \*\*генератор и дискриминатор\*\* учитывают \*\*дополнительные условия\*\* (класс изображения, текстовое описание, стиль и т. д.). Это позволяет управлять процессом генерации, создавая данные с заданными свойствами.

---

## \*\*1. Основные отличия cGAN от обычного GAN\*\*

| Обычный GAN | Conditional GAN (cGAN) |

|-------------|-----------------------|

| Генератор принимает \*\*только шум\*\* \( z \). | Генератор принимает \*\*шум + условие\*\* \( y \) (например, класс изображения). |

| Дискриминатор оценивает \*\*только реальность данных\*\*. | Дискриминатор проверяет \*\*соответствие данных условию\*\* \( y \). |

| Генерирует данные \*\*без контроля\*\*. | Генерирует данные \*\*с учётом заданных параметров\*\*. |

---

## \*\*2. Архитектура cGAN\*\*

### \*\*1. Генератор\*\*

- \*\*Вход\*\*:

- Случайный шум \( z \).

- Условие \( y \) (например, one-hot вектор класса).

- \*\*Обработка\*\*:

- Условие \( y \) объединяется с \( z \) (конкатенация или проекция).

- Генератор преобразует \( (z, y) \) в данные (например, изображение).

### \*\*2. Дискриминатор\*\*

- \*\*Вход\*\*:

- Данные \( x \) (реальные или сгенерированные).

- Условие \( y \).

- \*\*Выход\*\*:

- Вероятность того, что данные \*\*реальные и соответствуют условию\*\* \( y \).

---

## \*\*3. Математическая формулировка\*\*

Цель cGAN — оптимизация \*\*условной минимаксной игры\*\*:

\[

\min\_G \max\_D \mathbb{E}\_{x \sim p\_{\text{data}}}[\log D(x|y)] + \mathbb{E}\_{z \sim p\_z}[\log (1 - D(G(z|y)|y)]

\]

где:

- \( D(x|y) \) — вероятность того, что \( x \) реальное при условии \( y \),

- \( G(z|y) \) — сгенерированные данные для условия \( y \).

---

## \*\*4. Реализация cGAN на Python (Keras)\*\*

### \*\*1. Генератор с условием\*\*

```python

from keras.layers import Input, Embedding, Concatenate, Dense

# Условие (например, класс изображения)

label = Input(shape=(1,))

embedding = Embedding(input\_dim=10, output\_dim=50)(label) # 10 классов

flatten = Flatten()(embedding)

# Шум

noise = Input(shape=(100,))

# Объединение шума и условия

combined = Concatenate()([noise, flatten])

# Генератор

generator\_output = Dense(128, activation='relu')(combined)

generator\_output = Dense(784, activation='tanh')(generator\_output) # Пример для MNIST (28x28)

generator = Model(inputs=[noise, label], outputs=generator\_output)

```

### \*\*2. Дискриминатор с условием\*\*

```python

# Условие

label = Input(shape=(1,))

embedding = Embedding(input\_dim=10, output\_dim=50)(label)

flatten = Flatten()(embedding)

# Изображение (реальное или сгенерированное)

img = Input(shape=(784,)) # 28x28

# Объединение изображения и условия

combined = Concatenate()([img, flatten])

# Дискриминатор

discriminator\_output = Dense(128, activation='relu')(combined)

discriminator\_output = Dense(1, activation='sigmoid')(discriminator\_output)

discriminator = Model(inputs=[img, label], outputs=discriminator\_output)

```

### \*\*3. Обучение cGAN\*\*

```python

# Объединённая модель (генератор + дискриминатор)

discriminator.trainable = False

noise = Input(shape=(100,))

label = Input(shape=(1,))

fake\_img = generator([noise, label])

validity = discriminator([fake\_img, label])

gan = Model(inputs=[noise, label], outputs=validity)

gan.compile(loss='binary\_crossentropy', optimizer='adam')

# Цикл обучения

for epoch in range(epochs):

for batch in range(num\_batches):

# 1. Обучаем дискриминатор

real\_imgs, real\_labels = get\_real\_batch()

noise = np.random.normal(0, 1, (batch\_size, 100))

fake\_imgs = generator.predict([noise, real\_labels])

d\_loss\_real = discriminator.train\_on\_batch([real\_imgs, real\_labels], np.ones(batch\_size))

d\_loss\_fake = discriminator.train\_on\_batch([fake\_imgs, real\_labels], np.zeros(batch\_size))

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

# 2. Обучаем генератор

noise = np.random.normal(0, 1, (batch\_size, 100))

sampled\_labels = np.random.randint(0, 10, (batch\_size, 1))

g\_loss = gan.train\_on\_batch([noise, sampled\_labels], np.ones(batch\_size))

```

---

## \*\*5. Примеры применения cGAN\*\*

### \*\*1. Генерация изображений по классу\*\*

- Например, создание определённой цифры в MNIST или объекта в CIFAR-10.

### \*\*2. Текст → Изображение (Text-to-Image)\*\*

- Модели типа \*\*StackGAN\*\* генерируют изображения по текстовому описанию.

### \*\*3. Стилизация изображений\*\*

- Например, преобразование фото в стиль Ван Гога с указанием стиля.

---

## \*\*6. Проблемы cGAN\*\*

1. \*\*Несоответствие условий\*\*: Дискриминатор может игнорировать условие \( y \).

2. \*\*Сложность обучения\*\*: Требуется чёткое соответствие между условиями и данными.

3. \*\*Баланс между качеством и контролем\*\*: Условия могут ухудшить реалистичность.

---

## \*\*7. Развитие cGAN\*\*

- \*\*AC-GAN (Auxiliary Classifier GAN)\*\*: Дискриминатор дополнительно предсказывает класс.

- \*\*InfoGAN\*\*: Автоматическое выделение интерпретируемых условий.

- \*\*StyleGAN\*\*: Контроль стиля через промежуточные условия.

---

## \*\*8. Вывод\*\*

- \*\*cGAN\*\* — это GAN с дополнительными условиями, позволяющими управлять генерацией.

- \*\*Условия\*\* могут быть: классы, текст, стиль и т. д.

- \*\*Архитектура\*\*: Генератор и дискриминатор принимают \( y \) как вход.

- \*\*Применение\*\*: Генерация изображений по описанию, стилизация, контролируемый синтез.

\*\*Пример ответа на экзамене:\*\*

> \*"Conditional GAN (cGAN) — это GAN, в котором генератор и дискриминатор принимают дополнительное условие (например, класс изображения). Генератор создаёт данные, соответствующие условию, а дискриминатор проверяет их реалистичность и соответствие. Применяется для управляемой генерации изображений, текста и стилизации."\*

Вопрос № 44 Основные принципе алгоритма Q-learning.

### \*\*Основные принципы алгоритма Q-learning\*\*

\*\*Q-learning\*\* — это алгоритм \*\*обучения с подкреплением (Reinforcement Learning, RL)\*\*, который позволяет агенту находить оптимальную стратегию поведения в \*\*Марковском процессе принятия решений (MDP)\*\* без знания модели среды.

---

## \*\*1. Ключевые компоненты Q-learning\*\*

### \*\*1. Марковский процесс принятия решений (MDP)\*\*

- \*\*Состояния (States, \( s \))\*\*: Описание ситуации, в которой находится агент.

- \*\*Действия (Actions, \( a \))\*\*: Что может сделать агент в каждом состоянии.

- \*\*Награда (Reward, \( r \))\*\*: Мгновенная "польза" от перехода в новое состояние.

- \*\*Q-функция (Q-value, \( Q(s, a) \))\*\*: Ожидаемая суммарная награда за действие \( a \) в состоянии \( s \).

### \*\*2. Q-функция\*\*

Функция \( Q(s, a) \) оценивает, насколько "хорошо" выполнить действие \( a \) в состоянии \( s \).

\*\*Оптимальная Q-функция\*\* \( Q^\*(s, a) \) удовлетворяет \*\*уравнению Беллмана\*\*:

\[

Q^\*(s, a) = \mathbb{E}\left[ r + \gamma \max\_{a'} Q^\*(s', a') \right]

\]

где:

- \( s' \) — новое состояние,

- \( \gamma \) — коэффициент дисконтирования (обычно \( 0 \leq \gamma < 1 \)).

---

## \*\*2. Алгоритм Q-learning\*\*

### \*\*1. Инициализация\*\*

- Создаётся таблица \( Q(s, a) \) (изначально заполнена нулями или случайными значениями).

### \*\*2. Обновление Q-функции\*\*

На каждом шаге агент:

1. Выбирает действие \( a \) в состоянии \( s \) (например, через \*\*ε-жадную стратегию\*\*).

2. Получает награду \( r \) и переходит в состояние \( s' \).

3. Обновляет \( Q(s, a) \) по правилу:

\[

Q(s, a) \leftarrow Q(s, a) + \alpha \left[ r + \gamma \max\_{a'} Q(s', a') - Q(s, a) \right]

\]

где:

- \( \alpha \) — скорость обучения (learning rate),

- \( r + \gamma \max\_{a'} Q(s', a') \) — целевое значение (TD-target),

- \( Q(s, a) \) — текущее значение.

### \*\*3. Стратегия выбора действия (ε-жадная)\*\*

- С вероятностью \( \varepsilon \) выбирается \*\*случайное действие\*\* (исследование).

- С вероятностью \( 1 - \varepsilon \) выбирается \*\*жадное действие\*\* \( \arg\max\_a Q(s, a) \) (использование).

---

## \*\*3. Псевдокод Q-learning\*\*

```python

Initialize Q(s, a) arbitrarily (e.g., zeros)

for each episode:

Initialize state s

while s is not terminal:

Choose action a from s using ε-greedy policy

Take action a, observe reward r and next state s'

Update Q(s, a):

Q(s, a) = Q(s, a) + α \* [r + γ \* max\_a' Q(s', a') - Q(s, a)]

s = s'

Decrease ε (exploration rate)

```

---

## \*\*4. Пример реализации на Python\*\*

```python

import numpy as np

# Параметры

num\_states = 6

num\_actions = 2

alpha = 0.1

gamma = 0.9

epsilon = 0.1

episodes = 1000

# Инициализация Q-таблицы

Q = np.zeros((num\_states, num\_actions))

# Процесс обучения

for episode in range(episodes):

s = np.random.randint(0, num\_states) # Начальное состояние

while True:

# ε-жадный выбор действия

if np.random.rand() < epsilon:

a = np.random.randint(0, num\_actions)

else:

a = np.argmax(Q[s, :])

# Симуляция среды (пример: переход в s' и получение r)

s\_new = (s + 1) % num\_states # Простая среда: цикл по состояниям

r = 1 if s\_new == num\_states - 1 else 0 # Награда только в последнем состоянии

# Обновление Q-функции

Q[s, a] = Q[s, a] + alpha \* (r + gamma \* np.max(Q[s\_new, :]) - Q[s, a])

if s\_new == num\_states - 1:

break # Эпизод завершён

s = s\_new

```

---

## \*\*5. Важные особенности\*\*

### \*\*1. Off-policy алгоритм\*\*

- Q-learning обучается на \*\*оптимальной политике\*\* (через \( \max\_a Q(s', a') \)), но использует \*\*ε-жадную стратегию\*\* для исследования.

### \*\*2. Сходимость\*\*

- Q-learning гарантированно сходится к оптимальной политике, если:

- Все пары \( (s, a) \) посещаются бесконечно много раз.

- Скорость обучения \( \alpha \) уменьшается со временем.

### \*\*3. Проблемы\*\*

- \*\*Проклятие размерности\*\*: Для больших пространств состояний Q-таблица становится неэффективной (решение: \*\*Deep Q-Networks, DQN\*\*).

- \*\*Переоценка Q-значений\*\*: Из-за \( \max \) оператора (решение: \*\*Double Q-learning\*\*).

---

## \*\*6. Применение\*\*

- \*\*Игры\*\* (например, обучение бота для игры в шахматы).

- \*\*Робототехника\*\* (оптимизация движений).

- \*\*Управление ресурсами\*\* (например, беспилотные такси).

---

## \*\*7. Вывод\*\*

- \*\*Q-learning\*\* — это алгоритм обучения с подкреплением, основанный на обновлении Q-функции.

- \*\*Использует уравнение Беллмана\*\* для оценки будущих наград.

- \*\*Off-policy\*\*: Обучается на оптимальной политике, но исследует через ε-жадную стратегию.

- \*\*Применяется\*\* в играх, робототехнике и управлении.

\*\*Пример ответа на экзамене:\*\*

> \*"Q-learning — это off-policy алгоритм обучения с подкреплением, который находит оптимальную стратегию через итеративное обновление Q-функции. Основная формула: \( Q(s, a) = Q(s, a) + \alpha [r + \gamma \max\_{a'} Q(s', a') - Q(s, a)] \). Для исследования используется ε-жадная стратегия. Применяется в играх, робототехнике и управлении."\*

Вопрос № 45 Основные принципе алгоритма Police Gradient.

### \*\*Основные принципы алгоритма Policy Gradient\*\*

\*\*Policy Gradient (PG)\*\* — это метод \*\*обучения с подкреплением (Reinforcement Learning, RL)\*\*, который \*\*оптимизирует параметризованную политику напрямую\*\*, без использования таблиц Q-значений (как в Q-learning).

---

## \*\*1. Ключевые идеи Policy Gradient\*\*

### \*\*1. Параметризованная политика\*\*

- Политика \( \pi\_\theta(a|s) \) — это нейронная сеть (или другая модель), которая \*\*предсказывает вероятность выбора действия\*\* \( a \) в состоянии \( s \).

- Параметры \( \theta \) обновляются так, чтобы \*\*максимизировать ожидаемую награду\*\*.

### \*\*2. Цель обучения\*\*

Максимизировать \*\*ожидаемую суммарную дисконтированную награду\*\*:

\[

J(\theta) = \mathbb{E}\_{\tau \sim \pi\_\theta} \left[ \sum\_{t=0}^T \gamma^t r\_t \right]

\]

где:

- \( \tau = (s\_0, a\_0, r\_0, s\_1, a\_1, r\_1, \dots) \) — траектория,

- \( \gamma \) — коэффициент дисконтирования.

### \*\*3. Градиент политики (Policy Gradient Theorem)\*\*

Основная формула для обновления параметров:

\[

\nabla\_\theta J(\theta) = \mathbb{E}\_{\tau \sim \pi\_\theta} \left[ \sum\_{t=0}^T \nabla\_\theta \log \pi\_\theta(a\_t|s\_t) \cdot G\_t \right]

\]

где:

- \( G\_t = \sum\_{k=t}^T \gamma^{k-t} r\_k \) — \*\*дисконтированная кумулятивная награда\*\* с шага \( t \).

---

## \*\*2. Алгоритмы Policy Gradient\*\*

### \*\*1. REINFORCE (Ванильный Policy Gradient)\*\*

- \*\*Шаг 1\*\*: Сэмплируем траекторию \( \tau \) с текущей политикой \( \pi\_\theta \).

- \*\*Шаг 2\*\*: Для каждого шага \( t \) вычисляем \( G\_t \).

- \*\*Шаг 3\*\*: Обновляем параметры:

\[

\theta \leftarrow \theta + \alpha \sum\_{t=0}^T \nabla\_\theta \log \pi\_\theta(a\_t|s\_t) \cdot G\_t

\]

где \( \alpha \) — скорость обучения.

\*\*Недостатки\*\*:

- Высокая дисперсия градиентов (из-за случайности \( G\_t \)).

### \*\*2. Снижение дисперсии (Baseline, Advantage)\*\*

- \*\*Базлайн (Baseline)\*\*: Вычитаем среднюю награду \( b(s\_t) \), чтобы уменьшить дисперсию:

\[

\nabla\_\theta J(\theta) = \mathbb{E} \left[ \sum\_{t=0}^T \nabla\_\theta \log \pi\_\theta(a\_t|s\_t) \cdot (G\_t - b(s\_t)) \right]

\]

- \*\*Advantage-функция\*\*: \( A(s\_t, a\_t) = Q(s\_t, a\_t) - V(s\_t) \).

### \*\*3. Актор-Критик (Actor-Critic)\*\*

- \*\*Актор (Actor)\*\*: Политика \( \pi\_\theta(a|s) \), которая выбирает действия.

- \*\*Критик (Critic)\*\*: Оценивает \( V(s) \) или \( Q(s, a) \), чтобы снизить дисперсию.

- Обновление:

\[

\theta \leftarrow \theta + \alpha \nabla\_\theta \log \pi\_\theta(a\_t|s\_t) \cdot A(s\_t, a\_t)

\]

---

## \*\*3. Реализация REINFORCE на Python\*\*

```python

import numpy as np

import tensorflow as tf

# Определение политики (нейронная сеть)

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(4,)),

tf.keras.layers.Dense(2, activation='softmax') # 2 действия

])

optimizer = tf.keras.optimizers.Adam(learning\_rate=0.01)

gamma = 0.99

def train\_episode(env):

states, actions, rewards = [], [], []

s = env.reset()

# Сбор траектории

while True:

prob = model.predict(np.array([s]))[0]

a = np.random.choice(2, p=prob) # Сэмплируем действие

s\_new, r, done, \_ = env.step(a)

states.append(s)

actions.append(a)

rewards.append(r)

if done:

break

s = s\_new

# Вычисляем дисконтированные награды G\_t

G = 0

returns = []

for r in reversed(rewards):

G = r + gamma \* G

returns.insert(0, G)

# Обновляем параметры

with tf.GradientTape() as tape:

loss = 0

for s, a, G in zip(states, actions, returns):

prob = model(np.array([s]))[0]

log\_prob = tf.math.log(prob[a])

loss += -log\_prob \* G # Минимизируем -J(θ)

grads = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(grads, model.trainable\_variables))

```

---

## \*\*4. Плюсы и минусы Policy Gradient\*\*

✅ \*\*Преимущества\*\*:

- Работает в \*\*непрерывных\*\* пространствах действий (например, робототехника).

- Может обучать \*\*стохастические политики\*\* (например, в частично наблюдаемых средах).

- Прямая оптимизация целевой функции \( J(\theta) \).

❌ \*\*Недостатки\*\*:

- \*\*Высокая дисперсия\*\* градиентов (требует много эпизодов).

- \*\*Медленная сходимость\*\* по сравнению с Q-learning (но Actor-Critic решает эту проблему).

---

## \*\*5. Применение\*\*

- \*\*Игры\*\* (AlphaGo, Dota 2).

- \*\*Робототехника\*\* (управление движением).

- \*\*Финансы\*\* (оптимизация портфеля).

---

## \*\*6. Вывод\*\*

- \*\*Policy Gradient\*\* оптимизирует политику напрямую через градиентный подъём.

- \*\*Основная формула\*\*: \( \nabla\_\theta J(\theta) = \mathbb{E} [ \nabla\_\theta \log \pi\_\theta(a|s) \cdot G\_t ] \).

- \*\*Алгоритмы\*\*: REINFORCE, Actor-Critic.

- \*\*Плюсы\*\*: Работает с непрерывными действиями, минусы — высокая дисперсия.

\*\*Пример ответа на экзамене:\*\*

> \*"Policy Gradient — это метод RL, который оптимизирует параметризованную политику \( \pi\_\theta(a|s) \) через градиентный подъём. Основная формула обновления: \( \nabla\_\theta J(\theta) = \mathbb{E} [ \nabla\_\theta \log \pi\_\theta(a|s) \cdot G\_t ] \). Алгоритмы: REINFORCE (ванильный PG) и Actor-Critic (снижение дисперсии). Применяется в играх, робототехнике и финансах."\*

Вопрос № 46 Архитектура SequenceToSequence. Принцип построения.

### \*\*Архитектура Sequence-to-Sequence (Seq2Seq)\*\*

\*\*Sequence-to-Sequence (Seq2Seq)\*\* — это архитектура нейронных сетей, предназначенная для задач преобразования \*\*последовательностей переменной длины\*\* (например, машинный перевод, генерация текста, чат-боты).

---

## \*\*1. Основные компоненты Seq2Seq\*\*

### \*\*1. Энкодер (Encoder)\*\*

- \*\*Назначение\*\*: Преобразует входную последовательность (например, предложение на русском) в \*\*контекстный вектор\*\* (скрытое состояние).

- \*\*Архитектура\*\*:

- RNN (LSTM/GRU) или Transformer.

- Принимает входные токены пошагово и сохраняет скрытое состояние.

- \*\*Выход\*\*:

- Вектор контекста (последнее скрытое состояние энкодера).

### \*\*2. Декодер (Decoder)\*\*

- \*\*Назначение\*\*: Генерирует выходную последовательность (например, перевод на английский) на основе контекстного вектора.

- \*\*Архитектура\*\*:

- RNN (LSTM/GRU) или Transformer.

- На каждом шаге принимает \*\*предыдущий токен\*\* и \*\*скрытое состояние\*\*.

- \*\*Выход\*\*:

- Последовательность токенов (через softmax).

### \*\*3. Механизм внимания (Attention, опционально)\*\*

- Позволяет декодеру \*\*"фокусироваться"\*\* на разных частях входной последовательности.

- Улучшает качество для длинных последовательностей.

---

## \*\*2. Принцип работы Seq2Seq\*\*

### \*\*1. Фаза энкодера\*\*

- Входная последовательность \( X = (x\_1, x\_2, ..., x\_n) \) подаётся в энкодер.

- Энкoder обновляет скрытое состояние \( h\_t \):

\[

h\_t = \text{RNN}(x\_t, h\_{t-1})

\]

- Контекстный вектор \( c = h\_n \) (последнее состояние).

### \*\*2. Фаза декодера\*\*

- Декодер начинает с \*\*специального токена\*\* `<SOS>` (Start of Sequence).

- На каждом шаге \( t \):

- Принимает предыдущий токен \( y\_{t-1} \) и скрытое состояние \( s\_{t-1} \).

- Генерирует новый токен \( y\_t \):

\[

s\_t = \text{RNN}(y\_{t-1}, s\_{t-1}), \quad y\_t = \text{softmax}(W s\_t)

\]

- Процесс продолжается до токена `<EOS>` (End of Sequence).

### \*\*3. Attention (если используется)\*\*

- Декодер \*\*взвешивает\*\* скрытые состояния энкодера:

\[

c\_t = \sum\_{i=1}^n \alpha\_{ti} h\_i, \quad \alpha\_{ti} = \text{softmax}(\text{score}(s\_{t-1}, h\_i))

\]

- Где \( \alpha\_{ti} \) — вес внимания для \( h\_i \) на шаге \( t \).

---

## \*\*3. Реализация Seq2Seq с LSTM в PyTorch\*\*

### \*\*1. Энкодер\*\*

```python

import torch

import torch.nn as nn

class Encoder(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size):

super().\_\_init\_\_()

self.embedding = nn.Embedding(input\_size, hidden\_size)

self.lstm = nn.LSTM(hidden\_size, hidden\_size)

def forward(self, x):

# x: (seq\_len, batch\_size)

embedded = self.embedding(x) # (seq\_len, batch\_size, hidden\_size)

outputs, (hidden, cell) = self.lstm(embedded)

return hidden, cell

```

### \*\*2. Декодер\*\*

```python

class Decoder(nn.Module):

def \_\_init\_\_(self, output\_size, hidden\_size):

super().\_\_init\_\_()

self.embedding = nn.Embedding(output\_size, hidden\_size)

self.lstm = nn.LSTM(hidden\_size, hidden\_size)

self.fc = nn.Linear(hidden\_size, output\_size)

def forward(self, x, hidden, cell):

# x: (batch\_size)

x = x.unsqueeze(0) # (1, batch\_size)

embedded = self.embedding(x) # (1, batch\_size, hidden\_size)

output, (hidden, cell) = self.lstm(embedded, (hidden, cell))

prediction = self.fc(output.squeeze(0))

return prediction, hidden, cell

```

### \*\*3. Объединение в Seq2Seq\*\*

```python

class Seq2Seq(nn.Module):

def \_\_init\_\_(self, encoder, decoder):

super().\_\_init\_\_()

self.encoder = encoder

self.decoder = decoder

def forward(self, source, target, teacher\_forcing\_ratio=0.5):

# source: (src\_len, batch\_size)

# target: (trg\_len, batch\_size)

trg\_len, batch\_size = target.shape

outputs = torch.zeros(trg\_len, batch\_size, self.decoder.output\_size)

hidden, cell = self.encoder(source)

x = target[0] # Первый токен — <SOS>

for t in range(1, trg\_len):

output, hidden, cell = self.decoder(x, hidden, cell)

outputs[t] = output

x = target[t] if torch.rand(1) < teacher\_forcing\_ratio else output.argmax(1)

return outputs

```

---

## \*\*4. Обучение Seq2Seq\*\*

### \*\*1. Teacher Forcing\*\*

- На обучении декодеру подаётся \*\*правильный предыдущий токен\*\* (а не его собственный вывод).

- Ускоряет сходимость, но может привести к нестабильности.

### \*\*2. Потери (Loss Function)\*\*

- \*\*Cross-Entropy Loss\*\* между предсказанными и истинными токенами.

### \*\*3. Инференс (Генерация)\*\*

- На каждом шаге выбирается токен с \*\*максимальной вероятностью\*\* (жадный поиск) или через \*\*Beam Search\*\*.

---

## \*\*5. Плюсы и минусы Seq2Seq\*\*

✅ \*\*Преимущества\*\*:

- Работает с последовательностями \*\*переменной длины\*\*.

- Гибкость (можно использовать RNN, Transformer).

- Поддержка \*\*механизма внимания\*\* для сложных задач.

❌ \*\*Недостатки\*\*:

- \*\*Проблема длинных последовательностей\*\* (без внимания качество падает).

- \*\*Трудности с обучением\*\* (например, исчезающие градиенты в RNN).

---

## \*\*6. Применение\*\*

- \*\*Машинный перевод\*\* (Google Translate).

- \*\*Генерация текста\*\* (чат-боты, автодополнение).

- \*\*Синтез речи\*\* (Text-to-Speech).

---

## \*\*7. Вывод\*\*

- \*\*Seq2Seq\*\* состоит из \*\*энкодера\*\* (кодирует вход) и \*\*декодера\*\* (генерирует выход).

- \*\*Attention\*\* улучшает качество для длинных последовательностей.

- \*\*Реализация\*\*: RNN (LSTM/GRU) или Transformer.

- \*\*Применение\*\*: перевод, генерация текста, TTS.

\*\*Пример ответа на экзамене:\*\*

> \*"Seq2Seq — это архитектура для задач преобразования последовательностей, состоящая из энкодера (кодирует вход в контекстный вектор) и декодера (генерирует выход). Может использовать механизм внимания для улучшения качества. Применяется в машинном переводе, чат-ботах и синтезе речи."\*

Вопрос № 47 Сети с вниманием. Основные преимущества по сравнению с моделью seq2seq.

### \*\*Сети с вниманием (Attention Mechanisms): основные преимущества перед классической Seq2Seq\*\*

\*\*Классическая Seq2Seq\*\* использует фиксированный контекстный вектор (последнее скрытое состояние энкодера), что ограничивает её способность обрабатывать длинные и сложные последовательности.

\*\*Механизм внимания\*\* решает эту проблему, позволяя декодеру динамически "фокусироваться" на разных частях входной последовательности.

---

## \*\*1. Ключевые проблемы классической Seq2Seq\*\*

1. \*\*Информационное "бутылочное горлышко"\*\*

- Весь смысл входной последовательности кодируется в \*\*единственный вектор\*\* (последнее состояние энкодера).

- Для длинных текстов это приводит к потере информации.

2. \*\*Трудности с длинными последовательностями\*\*

- RNN/LSTM страдают от \*\*исчезающих градиентов\*\*, что ухудшает качество для длинных входов.

3. \*\*Жёсткая зависимость от порядка\*\*

- Декодер не может "заглядывать" назад в нужные части входных данных.

---

## \*\*2. Как работает механизм внимания?\*\*

### \*\*Основная идея\*\*

На каждом шаге декодера:

1. Вычисляются \*\*веса внимания\*\* (attention weights) для всех скрытых состояний энкодера.

2. Создаётся \*\*взвешенная сумма скрытых состояний\*\* — контекстный вектор \( c\_t \), уникальный для каждого шага декодера.

### \*\*Формулы внимания\*\*

\[

\begin{aligned}

e\_{ti} &= \text{score}(s\_{t-1}, h\_i) \quad \text{(энергия внимания)}, \\

\alpha\_{ti} &= \frac{\exp(e\_{ti})}{\sum\_{j=1}^n \exp(e\_{tj})} \quad \text{(softmax)}, \\

c\_t &= \sum\_{i=1}^n \alpha\_{ti} h\_i \quad \text{(контекстный вектор)}.

\end{aligned}

\]

Где:

- \( h\_i \) — скрытые состояния энкодера,

- \( s\_{t-1} \) — скрытое состояние декодера на предыдущем шаге,

- \( \alpha\_{ti} \) — вес внимания для \( h\_i \) на шаге \( t \).

---

## \*\*3. Преимущества Attention перед Seq2Seq\*\*

| \*\*Критерий\*\* | \*\*Классическая Seq2Seq\*\* | \*\*Seq2Seq + Attention\*\* |

|----------------------------|--------------------------------------------------|------------------------------------------------|

| \*\*Обработка длинных последовательностей\*\* | Плохая (теряется информация) | Отличная (динамический доступ к нужным частям входа) |

| \*\*Качество перевода/генерации\*\* | Среднее (зависит от сжатия в один вектор) | Высокое (учёт контекста для каждого токена) |

| \*\*Интерпретируемость\*\* | Низкая (чёрный ящик) | Высокая (можно визуализировать веса внимания) |

| \*\*Вычислительная сложность\*\* | Низкая (один проход энкодера) | Выше (расчёт внимания на каждом шаге декодера) |

### \*\*Конкретные преимущества Attention\*\*

1. \*\*Динамический контекст\*\*

- Каждый токен декодера получает свой контекстный вектор \( c\_t \), а не один общий.

2. \*\*Устранение "бутылочного горлышка"\*\*

- Нет потери информации, так как используются \*\*все\*\* скрытые состояния энкодера.

3. \*\*Улучшенная работа с длинными последовательностями\*\*

- Модель может "вспоминать" ранние части входа (например, в машинном переводе).

4. \*\*Лучшая интерпретируемость\*\*

- Визуализация весов \( \alpha\_{ti} \) показывает, на какие слова "смотрит" модель (полезно для отладки).

---

## \*\*4. Пример: Attention в машинном переводе\*\*

- \*\*Вход\*\*: "Кот ест рыбу".

- \*\*Выход\*\*: "The cat eats fish".

- \*\*Визуализация внимания\*\*:

- При генерации слова "cat" веса сосредоточены на "Кот".

- При генерации "fish" — на "рыбу".

---

## \*\*5. Развитие механизмов внимания\*\*

1. \*\*Self-Attention (Transformer)\*\*

- Полностью заменяет RNN/LSTM.

- Позволяет обрабатывать все токены параллельно (например, BERT, GPT).

2. \*\*Multi-Head Attention\*\*

- Несколько независимых механизмов внимания для разных аспектов данных.

3. \*\*Кросс-модальное внимание\*\*

- Например, между текстом и изображениями (VQA, Image Captioning).

---

## \*\*6. Вывод\*\*

\*\*Главные преимущества Attention перед классической Seq2Seq\*\*:

1. \*\*Динамический доступ\*\* к нужным частям входа (нет потери информации).

2. \*\*Лучшая производительность\*\* на длинных последовательностях.

3. \*\*Интерпретируемость\*\* через визуализацию весов.

\*\*Пример ответа на экзамене\*\*:

> \*"Механизм внимания решает ключевые проблемы классической Seq2Seq: устраняет информационное бутылочное горлышко, улучшает обработку длинных последовательностей и повышает интерпретируемость. Attention позволяет декодеру динамически фокусироваться на разных частях входа через взвешенную сумму скрытых состояний энкодера. Это применяется в Transformer, машинном переводе и генерации текста."\*

Вопрос № 48 Обнаружение объектов. Принцип построения архитектуры YOLO.

### \*\*Обнаружение объектов: архитектура YOLO (You Only Look Once)\*\*

\*\*YOLO\*\* (You Only Look Once) — это алгоритм \*\*обнаружения объектов\*\* в реальном времени, который предсказывает \*\*класс и координаты объектов\*\* за один проход через нейронную сеть (single-stage detector).

---

## \*\*1. Основные принципы YOLO\*\*

### \*\*1. Идея "одного прохода" (Single-Stage Detection)\*\*

- Традиционные методы (R-CNN, Faster R-CNN) используют \*\*двухэтапный подход\*\*:

1. Поиск регионов (Region Proposal).

2. Классификация и уточнение координат.

- \*\*YOLO\*\* объединяет эти этапы:

- \*\*Одна нейронная сеть\*\* сразу предсказывает классы и bounding box’ы.

### \*\*2. Разделение изображения на сетку (Grid Cells)\*\*

- Изображение делится на \( S \times S \) клеток (например, \( 7 \times 7 \)).

- Каждая клетка отвечает за обнаружение объектов, \*\*центр которых попадает в неё\*\*.

### \*\*3. Предсказание bounding box’ов\*\*

Для каждой клетки сетки YOLO предсказывает:

- \*\*Координаты\*\* \( (x, y, w, h) \):

- \( (x, y) \) — центр относительно клетки,

- \( (w, h) \) — ширина и высота относительно всего изображения.

- \*\*Уверенность\*\* (confidence score): Вероятность наличия объекта в box’е.

- \*\*Класс объекта\*\* (например, "человек", "автомобиль").

---

## \*\*2. Архитектура YOLO (на примере YOLOv3)\*\*

### \*\*1. Backbone (Основа)\*\*

- \*\*Darknet-53\*\*: Глубокая CNN (похожа на ResNet), извлекает признаки.

- Использует \*\*остаточные связи (Residual Blocks)\*\* для стабилизации обучения.

### \*\*2. Neck (Шея)\*\*

- \*\*FPN (Feature Pyramid Network)\*\*: Объединяет признаки разных масштабов для обнаружения объектов разного размера.

### \*\*3. Head (Голова)\*\*

- \*\*Три уровня детекции\*\* (на разных масштабах):

- \*\*Крупные объекты\*\* (например, автомобили).

- \*\*Средние объекты\*\* (например, люди).

- \*\*Мелкие объекты\*\* (например, мобильные телефоны).

---

## \*\*3. Принцип работы YOLO\*\*

1. \*\*Входное изображение\*\* делится на сетку \( S \times S \).

2. \*\*Каждая клетка\*\* предсказывает:

- \( B \) bounding box’ов (обычно \( B = 3 \)).

- Для каждого box’а: координаты \( (x, y, w, h) \), уверенность и классы.

3. \*\*Фильтрация дубликатов\*\*: Non-Maximum Suppression (NMS) оставляет только релевантные box’ы.

### \*\*Формула предсказания\*\*

Для каждой клетки сетки:

\[

\text{Output} = S \times S \times [B \cdot (5 + C)]

\]

где:

- \( 5 \) — координаты \( (x, y, w, h) \) + уверенность,

- \( C \) — число классов.

---

## \*\*4. Преимущества YOLO\*\*

✅ \*\*Скорость\*\*: Работает в реальном времени (45–150 FPS в зависимости от версии).

✅ \*\*Простота\*\*: Одна нейронная сеть вместо каскада моделей.

✅ \*\*Хорошая точность\*\* (особенно YOLOv4/YOLOv7).

❌ \*\*Недостатки\*\*:

- Трудности с \*\*мелкими объектами\*\* (но решается в YOLOv3+).

- Меньшая точность, чем у двухэтапных детекторов (но быстрее).

---

## \*\*5. Сравнение версий YOLO\*\*

| Версия | Год | Особенности |

|--------|-----|------------|

| \*\*YOLOv1\*\* | 2016 | Базовая архитектура, низкая точность для мелких объектов. |

| \*\*YOLOv3\*\* | 2018 | Darknet-53, FPN, 3 уровня детекции. |

| \*\*YOLOv4\*\* | 2020 | CSPDarknet, PANet, мозаичная аугментация. |

| \*\*YOLOv7\*\* | 2022 | Рекордная скорость и точность. |

---

## \*\*6. Пример кода (YOLOv3 на PyTorch)\*\*

```python

import torch

from models import Darknet # Реализация Darknet из официального кода YOLO

# Загрузка модели

model = Darknet("yolov3.cfg")

model.load\_weights("yolov3.weights")

# Детекция на изображении

img = torch.randn(1, 3, 416, 416) # Вход: 416x416

predictions = model(img) # Тензор с предсказаниями

# Фильтрация NMS

from utils import non\_max\_suppression

detections = non\_max\_suppression(predictions, conf\_thres=0.5, iou\_thres=0.4)

```

---

## \*\*7. Применение YOLO\*\*

- \*\*Беспилотные автомобили\*\* (обнаружение пешеходов, машин).

- \*\*Видеонаблюдение\*\* (трекинг объектов).

- \*\*Медицина\*\* (анализ рентгеновских снимков).

---

## \*\*8. Вывод\*\*

- \*\*YOLO\*\* — это single-stage детектор, работающий в реальном времени.

- \*\*Принцип\*\*: Делит изображение на сетку, каждая клетка предсказывает box’ы и классы.

- \*\*Архитектура\*\*: Backbone (Darknet) + FPN + три уровня детекции.

- \*\*Плюсы\*\*: Скорость, простота. \*\*Минусы\*\*: Сложности с мелкими объектами.

\*\*Пример ответа на экзамене\*\*:

> \*"YOLO — это single-stage детектор объектов, который предсказывает bounding box’ы и классы за один проход сети. Изображение делится на сетку, и каждая клетка отвечает за объекты, центры которых в неё попадают. Архитектура включает Darknet (backbone), FPN (neck) и три уровня детекции (head). Ключевые преимущества — скорость и простота, но могут быть проблемы с мелкими объектами."\*

Вопрос № 49 Параметризация аудио. Основные фичи, извлекаемые из аудиофайла.

### \*\*Параметризация аудио: основные извлекаемые фичи\*\*

\*\*Параметризация аудио\*\* — это процесс преобразования звукового сигнала в набор числовых характеристик (фич), которые описывают его свойства. Эти фичи используются в задачах машинного обучения, таких как:

- Распознавание речи (\*\*ASR\*\*).

- Классификация жанров музыки.

- Обнаружение эмоций в голосе.

- Генерация звука (например, TTS).

---

## \*\*1. Основные группы аудио-фич\*\*

### \*\*1. Временные (Time-Domain) фичи\*\*

Извлекаются непосредственно из сырого аудиосигнала (без преобразования в частотную область).

- \*\*Энергия сигнала (Amplitude Envelope)\*\*:

- Среднеквадратичное значение амплитуды (RMS).

- Используется для детекции активности голоса (VAD).

- \*\*Zero-Crossing Rate (ZCR)\*\*:

- Сколько раз сигнал пересекает нулевую ось за секунду.

- Полезно для区分 речи и шума, определения ударных в музыке.

- \*\*Темп (Tempo) и ритм\*\*:

- Анализ периодичности сигнала (например, для классификации музыки).

### \*\*2. Частотные (Frequency-Domain) фичи\*\*

Получаются после преобразования Фурье (FFT) или вейвлет-преобразования.

- \*\*Спектрограмма\*\*:

- Визуальное представление спектра частот во времени.

- Используется в CNN для классификации звуков.

- \*\*Мел-кепстральные коэффициенты (MFCC)\*\* — \*\*главная фича для речи\*\*:

- 13–40 коэффициентов, имитирующих восприятие звука человеком.

- Шаги получения MFCC:

1. Преобразование Фурье (STFT).

2. Применение мел-фильтров (подчеркивает важные для речи частоты).

3. Логарифмирование и DCT (дискретное косинусное преобразование).

- \*\*Spectral Centroid, Bandwidth, Rolloff\*\*:

- \*\*Центроид\*\*: "Центр тяжести" спектра (показывает яркость звука).

- \*\*Rolloff\*\*: Частота, ниже которой сосредоточено 85% энергии.

### \*\*3. Хроматические фичи (для музыки)\*\*

- \*\*Chroma Features\*\*:

- Распределение энергии по 12 полутонам (октава-независимые).

- Используется для определения аккордов и тональности.

- \*\*Tonnetz\*\*:

- Гармоническая проекция нот на 2D-плоскость.

### \*\*4. Фичи высокого уровня\*\*

- \*\*Формаманты (для речи)\*\*:

- Пики в спектре, характерные для гласных звуков.

- \*\*Pitch (частота основного тона)\*\*:

- Определяется через автокорреляцию или cepstrum.

- \*\*Гармонические и перкуссивные компоненты\*\*:

- Разделение музыки на мелодию и ударные.

---

## \*\*2. Примеры извлечения фич (Python)\*\*

### \*\*1. MFCC с Librosa\*\*

```python

import librosa

y, sr = librosa.load("audio.wav", sr=16000) # Загрузка аудио (16 кГц)

mfcc = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=13) # 13 коэффициентов

print("MFCC shape:", mfcc.shape) # (13, кадры)

```

### \*\*2. Спектрограмма\*\*

```python

import matplotlib.pyplot as plt

D = librosa.stft(y) # STFT

S\_db = librosa.amplitude\_to\_db(abs(D), ref=np.max)

librosa.display.specshow(S\_db, sr=sr, x\_axis='time', y\_axis='hz')

plt.colorbar()

```

### \*\*3. ZCR и Energy\*\*

```python

zcr = librosa.feature.zero\_crossing\_rate(y)

rms = librosa.feature.rms(y=y)

```

---

## \*\*3. Применение фич в ML\*\*

- \*\*Для Нейросетей\*\*:

- \*\*CNN\*\*: Спектрограммы или MFCC как 2D-изображения.

- \*\*RNN/LSTM\*\*: Последовательности MFCC для речи/музыки.

- \*\*Для Классических моделей\*\*:

- SVM/Random Forest на статистиках фич (среднее, дисперсия MFCC).

---

## \*\*4. Проблемы и решения\*\*

- \*\*Фоновые шумы\*\*: Использование фильтров (например, спектральное вычитание).

- \*\*Разная длина аудио\*\*: Padding или обрезка до фиксированного размера.

- \*\*Выбор фич\*\*: Для речи — MFCC, для музыки — Chroma + Spectral.

---

## \*\*5. Вывод\*\*

\*\*Основные аудио-фичи\*\*:

1. \*\*Временные\*\*: Энергия, ZCR.

2. \*\*Частотные\*\*: MFCC, спектрограмма.

3. \*\*Хроматические\*\*: Chroma, Tonnetz (для музыки).

4. \*\*Высокоуровневые\*\*: Pitch, формаманты.

\*\*Пример ответа на экзамене\*\*:

> \*"Параметризация аудио включает извлечение временных (энергия, ZCR), частотных (MFCC, спектрограмма) и хроматических фич (Chroma). MFCC — ключевая фича для речи, получаемая через STFT, мел-фильтры и DCT. Для музыки важны Chroma и Tonnetz. Фичи используются в CNN, RNN и классических ML-моделях."\*

Вопрос № 50 Базовые принципы генетического алгоритма.

### \*\*Генетические алгоритмы (ГА): базовые принципы\*\*

\*\*Генетический алгоритм (ГА)\*\* — это эвристический метод оптимизации, вдохновленный \*\*биологической эволюцией\*\* (естественным отбором, мутациями и рекомбинацией). Он используется для решения сложных задач, где традиционные методы неэффективны.

---

## \*\*1. Ключевые компоненты генетического алгоритма\*\*

| Компонент | Описание | Аналог в природе |

|--------------------|--------------------------------------------------------------------------|--------------------------------|

| \*\*Особи (Individuals)\*\* | Потенциальные решения задачи (например, векторы параметров). | Организмы в популяции. |

| \*\*Популяция (Population)\*\* | Набор особей, которые эволюционируют. | Популяция живых существ. |

| \*\*Фитнес-функция (Fitness Function)\*\* | Функция, оценивающая качество решения (чем выше — тем лучше). | Приспособленность к среде. |

| \*\*Селекция (Selection)\*\* | Выбор лучших особей для "размножения" (например, турнирная селекция). | Естественный отбор. |

| \*\*Кроссовер (Crossover)\*\* | Обмен генами между родителями для создания потомков. | Половое размножение. |

| \*\*Мутация (Mutation)\*\* | Случайное изменение генов у потомков. | Генетические мутации. |

---

## \*\*2. Пошаговый алгоритм\*\*

1. \*\*Инициализация популяции\*\*:

- Создается начальная популяция случайных особей.

- Пример: Для задачи минимизации функции \( f(x) \), особь — это вектор \( x = [x\_1, x\_2, \dots, x\_n] \).

2. \*\*Оценка приспособленности\*\*:

- Для каждой особи вычисляется фитнес-функция \( f(x) \).

3. \*\*Селекция\*\*:

- Выбираются особи с наилучшими значениями фитнес-функции.

- Методы:

- \*\*Рулетка\*\* (вероятность выбора пропорциональна фитнесу).

- \*\*Турнирная\*\* (случайно выбираются \( k \) особей, побеждает лучшая).

4. \*\*Кроссовер (Рекомбинация)\*\*:

- Две родительские особи обмениваются генами, создавая потомков.

- Пример (одноточечный кроссовер):

- Родители: \( [A,B,C,D] \) и \( [a,b,c,d] \).

- Потомок: \( [A,B,c,d] \).

5. \*\*Мутация\*\*:

- Случайное изменение части генов потомков (например, инверсия бита).

- Пример: \( [A,B,c,d] \rightarrow [A,B,C,d] \).

6. \*\*Формирование новой популяции\*\*:

- Новое поколение состоит из лучших родителей и потомков.

7. \*\*Критерий остановки\*\*:

- Алгоритм завершается при:

- Достижении максимального числа поколений.

- Нахождении решения с достаточной точностью.

- Отсутствии улучшений несколько итераций.

---

## \*\*3. Пример: Минимизация функции\*\*

\*\*Задача\*\*: Найти минимум функции \( f(x) = x^2 \) (очевидный ответ: \( x = 0 \)).

### \*\*Код на Python\*\*

```python

import numpy as np

# Параметры ГА

POP\_SIZE = 50

GENE\_LENGTH = 1 # x — число

GENERATIONS = 100

MUTATION\_RATE = 0.1

# Фитнес-функция (чем меньше x^2, тем лучше)

def fitness(x):

return -x\*\*2 # Минимизируем x^2 (знак '-' для "приспособленности")

# Инициализация популяции

population = np.random.uniform(-10, 10, (POP\_SIZE, GENE\_LENGTH))

for gen in range(GENERATIONS):

# Оценка приспособленности

scores = np.array([fitness(x) for x in population])

# Селекция (турнирная)

parents = []

for \_ in range(POP\_SIZE):

candidates = np.random.choice(range(POP\_SIZE), size=3, replace=False)

winner = candidates[np.argmax(scores[candidates])]

parents.append(population[winner])

# Кроссовер (одноточечный)

offspring = []

for i in range(0, POP\_SIZE, 2):

parent1, parent2 = parents[i], parents[i+1]

crossover\_point = np.random.randint(0, GENE\_LENGTH)

child1 = np.concatenate([parent1[:crossover\_point], parent2[crossover\_point:]])

child2 = np.concatenate([parent2[:crossover\_point], parent1[crossover\_point:]])

offspring.extend([child1, child2])

# Мутация

for child in offspring:

if np.random.rand() < MUTATION\_RATE:

child += np.random.normal(0, 0.5) # Добавляем шум

# Новая популяция

population = np.array(offspring)

# Лучший результат в поколении

best\_x = population[np.argmax(scores)]

print(f"Поколение {gen}: Лучший x = {best\_x[0]:.4f}, f(x) = {best\_x[0]\*\*2:.4f}")

```

---

## \*\*4. Плюсы и минусы ГА\*\*

✅ \*\*Преимущества\*\*:

- Работает с \*\*недифференцируемыми\*\* и \*\*разрывными\*\* функциями.

- Не требует градиента (полезно для black-box оптимизации).

- Параллелизуем (можно обрабатывать популяцию одновременно).

❌ \*\*Недостатки\*\*:

- Медленная сходимость для простых задач.

- Требует настройки параметров (размер популяции, мутация и др.).

- Риск \*\*преждевременной сходимости\*\* (популяция становится однородной).

---

## \*\*5. Применение\*\*

- \*\*Оптимизация параметров\*\* нейронных сетей.

- \*\*Расписание задач\*\* (например, составление школьного расписания).

- \*\*Дизайн\*\* (оптимизация формы крыла самолёта).

- \*\*ИИ в играх\*\* (настройка стратегий NPC).

---

## \*\*6. Вывод\*\*

- ГА имитируют \*\*естественный отбор\*\*, используя селекцию, кроссовер и мутацию.

- \*\*Особи\*\* представляют решения, \*\*фитнес-функция\*\* оценивает их качество.

- \*\*Плюсы\*\*: Универсальность, работа с сложными функциями. \*\*Минусы\*\*: Вычислительная затратность.

\*\*Пример ответа на экзамене\*\*:

> \*"Генетический алгоритм — это метод оптимизации, основанный на принципах естественного отбора. Основные этапы: инициализация популяции, оценка фитнес-функции, селекция, кроссовер и мутация. Применяется для задач, где традиционные методы неэффективны, например, в настройке гиперпараметров ML-моделей или проектировании."\*

Вопрос № 51 Подбор гиперпараметров нейронной сети с помощью генетического алгоритма.

### \*\*Подбор гиперпараметров нейронной сети с помощью генетического алгоритма\*\*

\*\*Генетический алгоритм (ГА)\*\* — это мощный инструмент для автоматического поиска оптимальных \*\*гиперпараметров нейронной сети\*\* (таких как learning rate, количество слоёв, размер batch и др.). В отличие от случайного поиска или GridSearch, ГА эффективно исследует пространство параметров, используя принципы \*\*естественной эволюции\*\*.

---

## \*\*1. Основные шаги подбора гиперпараметров с ГА\*\*

### \*\*1. Кодирование особи (индивида)\*\*

Каждая \*\*особь\*\* в популяции представляет собой \*\*набор гиперпараметров\*\*, закодированных в виде:

- \*\*Битовой строки\*\* (для дискретных параметров).

- \*\*Вещественного вектора\*\* (для непрерывных параметров).

\*\*Пример особи\*\*:

```python

# Гиперпараметры модели:

individual = {

"learning\_rate": 0.001, # float

"batch\_size": 32, # int

"n\_layers": 3, # int

"hidden\_units": [64, 128] # список

}

```

### \*\*2. Фитнес-функция (Fitness Function)\*\*

Функция, которая оценивает качество нейросети с заданными гиперпараметрами:

- \*\*Метрика\*\*: Accuracy, F1-score, Loss на валидационной выборке.

- \*\*Ускорение\*\*: Можно использовать \*\*упрощённое обучение\*\* (меньше эпох).

\*\*Пример\*\*:

```python

def evaluate\_model(params):

model = build\_model(params) # Создаёт модель с гиперпараметрами params

history = model.fit(X\_train, y\_train, epochs=5, validation\_data=(X\_val, y\_val))

return history.history["val\_accuracy"][-1] # Возвращает accuracy на валидации

```

### \*\*3. Генетические операторы\*\*

- \*\*Селекция\*\*: Отбор лучших особей (например, топ-20% по accuracy).

- \*\*Кроссовер\*\*: Комбинирование гиперпараметров двух родителей.

- Пример: Усреднение learning rate.

- \*\*Мутация\*\*: Случайное изменение параметров.

- Пример: Изменение числа слоёв \( \pm 1 \).

### \*\*4. Алгоритм\*\*

1. \*\*Инициализация\*\*: Создание начальной популяции со случайными гиперпараметрами.

2. \*\*Оценка\*\*: Обучение и валидация каждой особи.

3. \*\*Селекция\*\*: Выбор лучших для "размножения".

4. \*\*Кроссовер + Мутация\*\*: Генерация нового поколения.

5. \*\*Повторение\*\* шагов 2–4 до достижения критерия остановки.

---

## \*\*2. Пример реализации на Python\*\*

### \*\*1. Кодирование особи\*\*

```python

import numpy as np

def create\_individual():

return {

"learning\_rate": 10 \*\* np.random.uniform(-4, -1), # от 0.0001 до 0.1

"batch\_size": np.random.choice([16, 32, 64]),

"n\_layers": np.random.randint(1, 5),

"hidden\_units": [np.random.choice([32, 64, 128]) for \_ in range(np.random.randint(1, 4))]

}

```

### \*\*2. Фитнес-функция\*\*

```python

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

def build\_model(params):

model = Sequential()

for units in params["hidden\_units"]:

model.add(Dense(units, activation="relu"))

model.add(Dense(10, activation="softmax")) # Пример для MNIST

model.compile(optimizer=Adam(learning\_rate=params["learning\_rate"]),

loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

return model

def fitness\_function(params):

model = build\_model(params)

history = model.fit(X\_train, y\_train, epochs=5, batch\_size=params["batch\_size"], verbose=0)

return history.history["val\_accuracy"][-1] # Возвращаем accuracy

```

### \*\*3. Генетические операторы\*\*

```python

def crossover(parent1, parent2):

child = {}

for key in parent1:

if np.random.rand() < 0.5:

child[key] = parent1[key]

else:

child[key] = parent2[key]

return child

def mutate(individual, mutation\_rate=0.1):

for key in individual:

if np.random.rand() < mutation\_rate:

if key == "learning\_rate":

individual[key] \*= np.random.uniform(0.9, 1.1) # Изменяем на 10%

elif key == "batch\_size":

individual[key] = np.random.choice([16, 32, 64])

return individual

```

### \*\*4. Основной цикл ГА\*\*

```python

population = [create\_individual() for \_ in range(20)] # Популяция из 20 особей

for generation in range(10): # 10 поколений

scores = [fitness\_function(ind) for ind in population]

print(f"Поколение {generation}, Лучшая accuracy: {max(scores):.4f}")

# Селекция (топ-5)

top\_indices = np.argsort(scores)[-5:]

parents = [population[i] for i in top\_indices]

# Генерация потомков

offspring = []

for \_ in range(15): # 15 новых особей

parent1, parent2 = np.random.choice(parents, size=2, replace=False)

child = crossover(parent1, parent2)

child = mutate(child)

offspring.append(child)

# Новая популяция = родители + потомки

population = parents + offspring

```

---

## \*\*3. Плюсы и минусы метода\*\*

✅ \*\*Преимущества\*\*:

- \*\*Глобальный поиск\*\*: Может найти нетривиальные комбинации параметров.

- \*\*Параллелизация\*\*: Каждую особь можно обучать на отдельном GPU/CPU.

- \*\*Гибкость\*\*: Работает с любыми типами гиперпараметров.

❌ \*\*Недостатки\*\*:

- \*\*Вычислительная стоимость\*\*: Требует множества обучений модели.

- \*\*Субоптимальность\*\*: Нет гарантии нахождения глобального оптимума.

---

## \*\*4. Оптимизации\*\*

- \*\*Transfer Learning\*\*: Использование предобученных моделей для ускорения оценки.

- \*\*Ранняя остановка\*\*: Прекращение обучения, если accuracy не растёт.

- \*\*Суррогатные модели\*\*: Предсказание fitness без полного обучения (например, Bayesian Optimization + ГА).

---

## \*\*5. Вывод\*\*

- \*\*ГА\*\* эффективен для подбора гиперпараметров, особенно в сложных пространствах.

- \*\*Особь\*\* кодирует набор параметров, \*\*фитнес-функция\*\* — качество модели.

- \*\*Селекция, кроссовер, мутация\*\* позволяют исследовать пространство параметров.

\*\*Пример ответа на экзамене\*\*:

> \*"Генетический алгоритм для подбора гиперпараметров нейросети кодирует каждый набор параметров как особь, оценивает её через фитнес-функцию (например, accuracy на валидации) и применяет селекцию, кроссовер и мутацию для генерации новых решений. Плюсы: глобальный поиск и гибкость. Минусы: высокая вычислительная стоимость."\*

Вопрос № 52 Распознавание речи. Метрика WER.

### \*\*Распознавание речи и метрика WER (Word Error Rate)\*\*

\*\*Распознавание речи (Automatic Speech Recognition, ASR)\*\* — это задача преобразования устной речи в текст. Для оценки качества работы ASR-систем используется метрика \*\*Word Error Rate (WER)\*\* — процент ошибочно распознанных слов.

---

## \*\*1. Что такое WER?\*\*

\*\*WER\*\* — это стандартная метрика в ASR, которая измеряет, насколько распознанный текст отличается от эталонного (оригинального).

Формула:

\[

WER = \frac{S + D + I}{N} \times 100\%

\]

где:

- \*\*S (Substitutions)\*\* — замены слов (например, "кот" → "код").

- \*\*D (Deletions)\*\* — пропущенные слова.

- \*\*I (Insertions)\*\* — лишние слова.

- \*\*N\*\* — общее число слов в эталонном тексте.

\*\*Пример\*\*:

| Эталонный текст | "кот ест рыбу" |

|----------------------|----------------|

| Распознанный текст | "код ест" |

| \*\*Ошибки\*\* | S=1 ("кот"→"код"), D=1 ("рыбу"), I=0 |

| \*\*WER\*\* | \( \frac{1 + 1 + 0}{3} \times 100\% = 66.7\% \) |

---

## \*\*2. Как вычисляется WER?\*\*

### \*\*Алгоритм (на основе расстояния Левенштейна)\*\*

1. \*\*Разбиение на слова\*\*:

- Эталон: \( R = ["кот", "ест", "рыбу"] \).

- Гипотеза (распознанный текст): \( H = ["код", "ест"] \).

2. \*\*Построение матрицы ошибок\*\*:

- Сравниваются слова из \( R \) и \( H \), подсчитываются S, D, I.

3. \*\*Расчёт WER\*\*:

- По формуле выше.

### \*\*Пример кода (Python)\*\*

```python

from jiwer import wer

reference = "кот ест рыбу"

hypothesis = "код ест"

error\_rate = wer(reference, hypothesis)

print(f"WER: {error\_rate \* 100:.2f}%") # Вывод: WER: 66.67%

```

---

## \*\*3. Особенности WER\*\*

### \*\*Преимущества\*\*

✅ \*\*Простота\*\*: Легко интерпретировать (чем меньше, тем лучше).

✅ \*\*Универсальность\*\*: Применима к любым ASR-системам.

### \*\*Недостатки\*\*

❌ \*\*Чувствительность к порядку слов\*\*:

- Фраза "рыбу ест кот" даст WER = 100%, хотя смысл тот же.

❌ \*\*Игнорирует семантику\*\*:

- Не учитывает схожесть значений слов (например, "кот" → "кошка").

### \*\*Дополнительные метрики\*\*

- \*\*CER (Character Error Rate)\*\*: Ошибки на уровне символов.

- \*\*Semantic WER\*\*: Учитывает смысловую близость слов.

---

## \*\*4. Как улучшить WER?\*\*

1. \*\*Использование языковой модели\*\*:

- GPT, BERT для коррекции распознанного текста.

2. \*\*Акустическая адаптация\*\*:

- Fine-tuning модели на конкретного диктора или шумы.

3. \*\*Постобработка\*\*:

- Исправление опечаток (например, библиотека `symspellpy`).

---

## \*\*5. Примеры WER в реальных системах\*\*

| Система | WER (%) | Условия |

|-----------------------|---------|-----------------------------|

| \*\*Google Speech API\*\* | 5–10 | Чистая речь (английский) |

| \*\*DeepSpeech\*\* | 10–15 | С шумами |

| \*\*Whisper (OpenAI)\*\* | 3–5 | Мультиязычные данные |

---

## \*\*6. Вывод\*\*

- \*\*WER\*\* — ключевая метрика для оценки ASR-систем.

- \*\*Формула\*\*: \( \frac{S + D + I}{N} \times 100\% \).

- \*\*Оптимизация\*\*: Через языковые модели и акустическую адаптацию.

\*\*Пример ответа на экзамене\*\*:

> \*"WER (Word Error Rate) — это метрика для оценки качества распознавания речи, вычисляемая как отношение суммы замен (S), удалений (D) и вставок (I) к общему числу слов в эталонном тексте (N). Недостатки: не учитывает семантику и порядок слов. Современные ASR-системы (например, Whisper) достигают WER 3–5% на чистой речи."\*

Вопрос № 53 Синтез речи. Конкатенативный и параметрический подходы. Библиотеки синтеза речи.

### \*\*Синтез речи: конкатенативный и параметрический подходы. Библиотеки\*\*

\*\*Синтез речи (Text-to-Speech, TTS)\*\* — это технология преобразования текста в естественно звучащую речь. Существует два основных подхода: \*\*конкатенативный\*\* и \*\*параметрический\*\*.

---

## \*\*1. Конкатенативный синтез\*\*

### \*\*Принцип работы\*\*

- Использует \*\*заранее записанные фрагменты речи\*\* (слова, слоги, фонемы).

- Фрагменты \*\*склеиваются\*\* (конкатенируются) для формирования фраз.

### \*\*Виды конкатенативного синтеза\*\*

1. \*\*На уровне слов\*\*:

- Запись частотных слов (например, "привет", "пока").

- Ограничение: неестественность при сложных фразах.

2. \*\*На уровне слогов/фонем\*\*:

- Более гибкий подход, но требует сложного склеивания.

### \*\*Плюсы и минусы\*\*

✅ \*\*Преимущества\*\*:

- Естественное звучание (если фразы записаны человеком).

- Низкие вычислительные затраты.

❌ \*\*Недостатки\*\*:

- Ограниченный словарь (невозможно синтезировать новые слова).

- "Роботизированный" звук на стыках фрагментов.

### \*\*Примеры систем\*\*

- \*\*Festival\*\* (старая система, поддерживает русский).

- \*\*Unit Selection\*\* (использует длинные фрагменты для лучшего качества).

---

## \*\*2. Параметрический синтез\*\*

### \*\*Принцип работы\*\*

- Речь генерируется \*\*с нуля\*\* на основе параметров:

- \*\*Мел-кепстральные коэффициенты (MFCC)\*\* + \*\*F0 (тон)\*\* + \*\*длительность\*\*.

- Использует \*\*статистические модели\*\* или \*\*нейросети\*\* (например, Tacotron, WaveNet).

### \*\*Методы\*\*

1. \*\*HMM-синтез (Hidden Markov Models)\*\*:

- Устаревший, но простой метод.

2. \*\*Нейросетевой синтез\*\*:

- \*\*Tacotron 2\*\* (преобразует текст в спектрограмму).

- \*\*WaveNet\*\* (генерирует сырой звук из параметров).

### \*\*Плюсы и минусы\*\*

✅ \*\*Преимущества\*\*:

- Может синтезировать \*\*любые слова\*\* (даже незнакомые).

- Плавная, почти человеческая речь (с современными моделями).

❌ \*\*Недостатки\*\*:

- Требует \*\*больших данных\*\* для обучения.

- Высокие вычислительные затраты.

### \*\*Примеры систем\*\*

- \*\*Google WaveNet\*\*, \*\*Amazon Polly\*\*.

- \*\*Модели на основе Transformer\*\* (например, FastSpeech).

---

## \*\*3. Современные библиотеки для TTS\*\*

| Библиотека | Подход | Языки | Особенности |

|----------------------|------------------|-------------|--------------------------------------|

| \*\*eSpeak\*\* | Формантный | Мультиязык | Роботизированный звук, но малый размер. |

| \*\*Festival\*\* | Конкатенативный | EN, RU | Гибкость, требует ручной настройки. |

| \*\*MaryTTS\*\* | HMM + Unit | EN, DE, RU | Открытый аналог коммерческих систем. |

| \*\*Tacotron 2\*\* | Нейросетевой | EN, RU (есть модели) | Высокое качество, требует GPU. |

| \*\*Coqui TTS\*\* | Нейросетевой | Мультиязык | Поддержка современных моделей (VITS, Glow-TTS). |

| \*\*RHVoice\*\* | Unit Selection | RU, EN | Оптимизирован для русского языка. |

---

## \*\*4. Сравнение подходов\*\*

| Критерий | Конкатенативный | Параметрический |

|------------------------|--------------------------|---------------------------|

| \*\*Естественность\*\* | Высокая (если фразы записаны) | Очень высокая (современные модели) |

| \*\*Гибкость\*\* | Низкая (ограниченный словарь) | Высокая (любые слова) |

| \*\*Вычислительная сложность\*\* | Низкая | Высокая |

| \*\*Пример\*\* | Festival, RHVoice | Tacotron 2, WaveNet |

---

## \*\*5. Как выбрать подход?\*\*

- \*\*Для ограниченного набора фраз\*\* (например, автоответчик) → \*\*конкатенативный\*\*.

- \*\*Для произвольного текста\*\* → \*\*параметрический (нейросетевой)\*\*.

\*\*Пример кода (Coqui TTS):\*\*

```python

from TTS.api import TTS

tts = TTS(model\_name="tts\_models/ru/vits", progress\_bar=False)

tts.tts\_to\_file(text="Привет, мир!", file\_path="output.wav")

```

---

## \*\*6. Вывод\*\*

- \*\*Конкатенативный синтез\*\* использует записанные фрагменты, подходит для статичных фраз.

- \*\*Параметрический синтез\*\* генерирует речь из параметров, идеален для динамического текста.

- \*\*Лучшие библиотеки\*\*: Tacotron 2, WaveNet (нейросети), RHVoice (русский язык).

\*\*Пример ответа на экзамене\*\*:

> \*"Конкатенативный синтез речи склеивает заранее записанные фрагменты, обеспечивая естественность, но ограничен словарём. Параметрический синтез (например, Tacotron 2) генерирует речь через нейросети, поддерживая любые тексты. Для русского языка популярны RHVoice и нейросетевые модели Coqui TTS."\*