

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**Федеральное государственное автономное образовательное учреждение**  
**высшего образования**  
**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Методические указания**

по выполнению лабораторных работ  
по дисциплине

**«Системы искусственного интеллекта»**

для направления подготовки

44.03.05 Педагогическое образование (с двумя профилями подготовки)

направленность (профиль):

География и биология

Ставрополь, 2026

## **СОДЕРЖАНИЕ**

**Лабораторная работа 1.** Новые информационные технологии и Системы искусственного интеллекта (СИИ)

**Лабораторная работа 2.** Искусственный интеллект: основные понятия и история возникновения

**Лабораторная работа 3.** Основные направления искусственного интеллекта

**Лабораторная работа 4.** Экспертные системы

**Лабораторная работа 5.** Модели представления знаний в экспертных системах

**Лабораторная работа 6.** Общение человека с системой ИИ

**Лабораторная работа 7.** Решение задач и искусственный интеллект

## ВВЕДЕНИЕ

Цель освоения дисциплины является овладение студентами основными методами теории интеллектуальных систем, приобретение навыков по использованию интеллектуальных систем, изучение основных методов представления знаний и моделирования рассуждений.

Задачи освоения дисциплины – помочь студентам овладеть навыками и знаниями в области искусственного интеллекта

### Наименование лабораторных работ

| № Темы дисциплины | Наименование тем дисциплины, их краткое содержание                           | Объем часов | Из них практическая подготовка, часов |
|-------------------|------------------------------------------------------------------------------|-------------|---------------------------------------|
| 5 семестр         |                                                                              |             |                                       |
| 1.                | Новые информационные технологии и<br>Системы искусственного интеллекта (СИИ) | 2           | 2                                     |
| 2.                | Искусственный интеллект:<br>основные понятия и<br>история возникновения      | 2           | 2                                     |
| 3.                | Основные направления<br>искусственного интеллекта                            | 2           | 2                                     |
| 4.                | Экспертные системы                                                           | 2           | 2                                     |
| 5.                | Модели представления знаний в<br>экспертных системах                         | 2           | 2                                     |
| 6.                | Общение человека с системой<br>ИИ                                            | 2           | 2                                     |
| 7.                | Решение задач и искусственный<br>интеллект                                   | 2           | 2                                     |
|                   | Итого за 5 семестр                                                           | 14          | 14                                    |
|                   | Итого                                                                        | 14          | 14                                    |

## СТРУКТУРА И СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

## ЛР №1. Новые информационные технологии и Системы искусственного интеллекта (СИИ)

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Традиционные средства программного обеспечения ЭВМ и системы ИИ.
2. Задачи работ в области ИИ.
3. Тест Тьюринга.
4. Моделирование окружающего мира и поведения человека.
5. Интеллектуальная деятельность человека и ИИ.
6. Основные школы психологии мышления.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

**Задание 3.**

Изучив литературные источники и материалы из сети Интернет, заполните следующую таблицу «Нормативно-правовые основы развития искусственного интеллекта в мире» (привести не менее 7 стран):

| п/п | Страна | Документ | Дата принятия | Сроки реализации (если имеются) | Цели и задачи |
|-----|--------|----------|---------------|---------------------------------|---------------|
| .   |        | 1.1.     |               |                                 |               |
|     |        | 1.2.     |               |                                 |               |
|     |        | ...      |               |                                 |               |
| .   | ...    |          |               |                                 |               |

**Задание 4.**

Изучив литературные источники и материалы из сети Интернет, заполните следующую таблицу «Примеры технологий ИИ в моей профессиональной сфере» (привести 5- 7 технологий):

| п/п | Название технологии | Разработчи к технологии | Время появления технологии | Описание возможностей технологии | Примеры использования технологии в профессиональной сфере |
|-----|---------------------|-------------------------|----------------------------|----------------------------------|-----------------------------------------------------------|
|     |                     |                         |                            |                                  |                                                           |
|     |                     |                         |                            |                                  |                                                           |

## ЛР №2. Искусственный интеллект: основные понятия и история возникновения

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Основные понятия систем искусственного интеллекта.
2. Общая характеристика задач решаемых методами искусственного интеллекта.
3. История развития искусственного интеллекта.
4. Современное состояние искусственного интеллекта.
5. Программное обеспечение работ по ИИ.

6. Экспериментальный и эволюционный характер разработок систем ИИ, требования к программному обеспечению.
7. Языки программирования для задач ИИ.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

**Задание 3.**

1. Изучить теоретическое введение.
2. Ответить на контрольные вопросы, приведенные в конце лабораторной работы.
3. Выполнить практические задания с использованием языка программирования Python.
4. Составить отчет по лабораторной работе. Требования к отчету:  
Отчёт предоставляется в электронном виде в текстовом редакторе MS Word. В отчёте указываются:
  - 1) Фамилия студента.
  - 2) Порядковый номер и название лабораторной работы.
  - 3) Цель работы.
  - 4) Ответы на контрольные вопросы.
  - 5) Описание решения выполненных заданий лабораторной работы, содержащее: а) формулировку задания; в) скриншот выполненного задания, содержащий фамилию студента.Защита лабораторной работы осуществляется по отчёту, представленному студентом и с демонстрацией задания на компьютере.

### **ЛР №3. Основные направления искусственного интеллекта**

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Классификация систем искусственного интеллекта.
2. Экспертные системы
3. Самообучающиеся системы.
4. Адаптивные системы.
5. Характеристики знаний.
6. Модели представления знаний.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

#### **Начало работы с Python**

Перейдите к следующей ячейке «In []», введите представленный ниже код и выполните его, щелкнув на кнопке запуска. В отношении инструкций, записанных на компьютерном языке, широко применяется термин код. Вместо щелчка на кнопке запуска кода можете воспользоваться комбинацией клавиш <Ctrl+Enter>, если вам этот способ более удобен.

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

В ответ компьютер должен просто вывести в окне фразу «Hello, World!»

```
In [2]: print("Hello, World!")  
Hello, World!
```

Ввод инструкции, предписывающей вывод фразы «Hello, World!», не привел к удалению предыдущей ячейки с содержащимися в ней собственной инструкцией и собственным выводом. Это средство оказывается очень полезным при поэтапном создании решений из нескольких частей.

Посмотрим, что произойдет при выполнении следующего кода, который демонстрирует одну ключевую идею. Введите и выполните этот код в новой ячейке. Если новая пустая ячейка не отображается в окне, щелкните на кнопке с изображением знака «плюс», после наведения на которую указателя мыши высвечивается подсказка Insert Cell Below (вставить ячейку снизу).

```
x = 10  
  
print(x)  
  
print(x+5)  
  
y = x+7  
  
print(y)  
  
print (z)
```

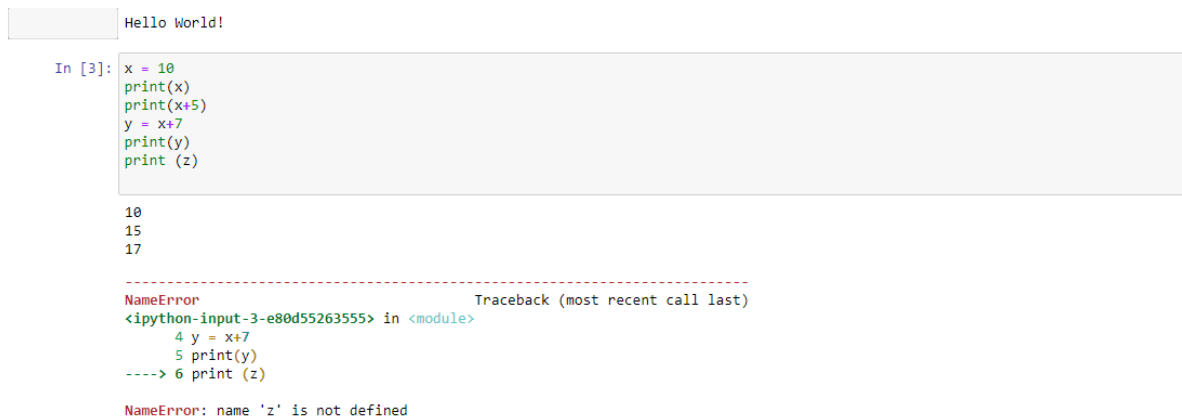
Первая строка, `x = 10`, выглядит как математическая запись, утверждающая, что `x` равно 10. В Python это означает, что в виртуальное хранилище под названием `x` заносится значение 10.

Значение «10» остается в хранилище до тех пор, пока в нем не возникнет необходимость. Инструкция `print (x)`, с которой вы уже сталкивались, выводит информации на экран. Она выдаст хранящееся в `x` значение, которое равно 10. Но почему будет выведено «10», а не «`x`»? Потому что Python всегда старается вычислить все, что только можно, а `x` можно вычислить и получить значение 10, которое и выводится. В следующей строке, которая содержит инструкцию `print (x+5)`, вычисляется выражение `x+5`, приводящее в конечном счете к значению `10+5`, или 15. Поэтому мы ожидаем, что на экран будет выведено «15».

Следующая строка с инструкцией `y=x+7` также должна быть понятной, если вспомнить, что Python стремится выполнить все возможные вычисления. В этой инструкции мы предписываем сохранить значение в новом хранилище, которое теперь называется `y`, но при этом возникает вопрос, а какое именно значение мы хотим сохранить? В инструкции указано выражение `x+7`, которое равно `10+7`, т.е. 17. Таким образом, именно это значение и будет сохранено в `y`, и следующая инструкция выведет его на экран.

А что произойдет со строкой `print (z)`, если мы не назначили `z` никакого значения, как это было сделано в случае `x` и `y`? Мы получим сообщение об ошибке, информирующее о некорректности предпринимаемых действий и пытающееся быть максимально полезным, чтобы можно было устранить ошибку. Сообщения об ошибках не всегда успешно справляются со своей задачей – оказать помощь пользователю (причем этот недостаток характерен для большинства языков программирования).

Результаты выполнения описанного кода, включая сообщение об ошибке «`name V is not defined`» (имя '`z`' не определено), можно увидеть на следующей иллюстрации.



```
Hello World!  
  
In [3]: x = 10  
        print(x)  
        print(x+5)  
        y = x+7  
        print(y)  
        print (z)  
  
10  
15  
17  
  
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-3-e80d55263555> in <module>  
      4 y = x+7  
      5 print(y)  
----> 6 print (z)  
  
NameError: name 'z' is not defined
```

Вышеупомянутые хранилища, обозначенные как *x* и *y*, и используемые для хранения таких значений, как 10 и 17, называют **переменными**. В языках программирования переменные используются для создания обобщенных наборов инструкций по аналогии с тем, как математики используют алгебраические символы наподобие «*x*» и «*y*» для формулировки утверждений общего характера.

### 3. Многократное выполнение команд

Компьютеры подходят для многократного выполнения однотипных задач – они не размышляют и производят вычисления намного быстрее, чем это удастся делать людям с помощью калькуляторов.

Посмотрим, как вывести квадраты первых десяти натуральных чисел, начиная с нуля: 0 в квадрате, 1 в квадрате, 2 в квадрате и т.д. В результате должен получиться следующий ряд чисел: 0, 1, 4, 9, 16, 25 и т.д.

Мы могли бы самостоятельно произвести все эти вычисления, а затем просто использовать инструкции вывода `print (0)`, `print (1)`, `print (4)` и т.д. Это сработает, но ведь наша цель заключается в том, чтобы всю вычислительную работу вместо нас выполнил компьютер. Также нам необходимо создать типовой набор инструкций, позволяющий выводить квадраты чисел в любом заданном диапазоне. Для этого нужно сначала рассмотреть несколько новых особенностей работы данного языка программирования.

Введите в очередную пустую ячейку следующий код:

```
list( range(10) )
```

В итоге получается список из десяти чисел от 0 до 9.

```
In [4]: list( range(10) )
Out[4]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [ ]: |
```

Список содержит числа от 0 до 9, а не от 1 до 10. Это не случайно. Компьютерный список начинается с 1, а не с 0. Упорядоченные списки широко используются в качестве счетчиков при многократном выполнении некоторых вычислений и, в частности, когда используются итеративные функции.

В данном случае мы опустили ключевое слово `print`, которое использовали при выводе фразы «Hello, World!». Использование ключевого слова `print` не является обязательным при работе с Python в интерактивном режиме, поскольку оболочка знает, что мы хотим увидеть результат введенной инструкции.

Распространенным способом многократного выполнения вычислений с помощью компьютера является использование так называемых циклов. Слово «цикл» правильно передает суть происходящего, когда некоторые действия повторяются снова и снова, возможно, даже бесконечное число раз. Вместо того чтобы приводить формальное определение цикла, гораздо полезнее рассмотреть конкретный простой пример. Введите и выполните в новой ячейке следующий код.

```
for n in range(10):
```

```
    print(n)
```

```
    pass
```

```
print("готово")
```

Здесь имеются три новых элемента, которые мы последовательно разберем. Первая строка содержит выражение `range(10)`, с которым вы уже знакомы. Оно создает список чисел от 0 до 9.

Конструкция `for n in` – это тот элемент, который создает цикл и заставляет выполнять некоторые действия для каждого числа из списка, организуя счетчик путем назначения текущего значения переменной `n`. Ранее мы уже обсуждали присваивание значений переменным, и здесь происходит то же самое: при первом проходе цикла выполняется присваивание `n=0`, а при последующих – присваивание `n=1`, `n=2` и так до тех пор, пока мы не дойдем до последнего элемента списка, для которого будет выполнено присваивание `n=9`.

Смысл инструкции `print(n)` в следующей строке, которая просто выводит текущее значение `n`, уже должен быть понятен и знаком. Но обратите внимание на отступ перед текстом `print(n)`. В Python отступы играют важную роль, поскольку намеренно используются для того, чтобы показать подчиненность одних инструкций другим, в данном случае циклу, созданному с помощью конструкции `for n in`. Инструкция `pass` сигнализирует о конце цикла, и следующая строка, записанная с использованием обычного отступа, не является частью цикла. Это означает, что мы ожидаем, что слово «готово» будет выведено только один раз, а не десять. Представленный ниже результат подтверждает наши ожидания.

```
In [5]: for n in range(10):
        print(n)
        pass
        print("готово")

0
1
2
3
4
5
6
7
8
9
готово

In [ ]: |
```

Теперь должно быть понятно, что для получения квадратов чисел следует выводить на экран значения `n * n`. В действительности мы можем поступить еще лучше и выводить фразы наподобие «Квадрат числа 3 равен 9». Заметьте, что в инструкции переменные не заключаются в кавычки и поэтому вычисляются.

```
for n in range(10):
```

```
    print("квадрат числа", n, "равен", n*n)
```

```
    pass
```

```
print("готово")
```

Результат приведен ниже.

```
In [18]: for n in range(10):
        print("квадрат числа", n, "равен", n*n)
        pass
        print("готово")

квадрат числа 0 равен 0
квадрат числа 1 равен 1
квадрат числа 2 равен 4
квадрат числа 3 равен 9
квадрат числа 4 равен 16
квадрат числа 5 равен 25
квадрат числа 6 равен 36
квадрат числа 7 равен 49
квадрат числа 8 равен 64
квадрат числа 9 равен 81
готово

In [ ]: |
```

Точно так же можно увеличить число итераций до пятидесяти или тысячи с помощью выражений `range(50)` или `range(1000)`. Попробуйте сделать это самостоятельно.

#### **4. Создание комментариев**

Рассмотрим, как задать комментарии в Python. Наберите приведенный ниже простой код.

# следующая инструкция выводит куб числа 2

```
print(2**3)
```

Первая строка начинается с символа решетки (#). Python игнорирует все строки, начинающиеся с этого символа. Однако эти строки не бесполезны: с их помощью мы можем оставлять в коде полезные комментарии, которые помогут понять его предназначение другим людям или даже нам самим, если мы обратимся к нему после долгого перерыва.

```
In [19]: # следующая инструкция выводит куб числа 2
print(2**3)
8
```

```
In [ ]: |
```

### **ЛР №4. Экспертные системы**

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Понятие экспертной системы
2. Структура экспертных систем
3. Этапы разработки экспертных систем.
4. Области применения ЭС.
5. Архитектура ЭС.
6. База знаний, механизмы вывода, подсистемы объяснения, общения, приобретения знаний ЭС.
7. Жизненный цикл экспертной системы.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

### **ЛР №5. Модели представления знаний в экспертных системах**

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Представление знаний в экспертных системах
2. Семантические сети
3. Фреймовая модель
4. Продукционная модель
5. Логическая модель
6. Методы поиска решений
7. Выбор метода поиска решений

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

## 1. Функции

Ранее мы работали с математическими функциями. Данные функции получают входные данные, выполняют некую работу и выдают результат. Эти функции действовали самостоятельно, и мы могли их многократно использовать.

Многие языки программирования, включая Python, упрощают создание повторно используемых инструкций. Подобно математическим функциям такие многократно используемые фрагменты кода, если они определены надлежащим образом, могут действовать автономно и обеспечивают создание более короткого элегантного кода.

Почему сокращается объем кода? Потому что вызывать функцию, обращаясь к ее имени, гораздо проще, чем каждый раз полностью записывать образующий эту функцию код.

Для задания функции необходимо четко определить, *какие входные данные ожидает функция* и *какой тип выходных данных* она выдает. Некоторые функции в качестве входных данных принимают только числа, и поэтому им нельзя передавать состоящие из букв слова.

Обратимся к конкретному примеру. Введите и выполните следующий код.

# функция, которая принимает два числа в качестве входных данных # и выводит их среднее значение

```
def среднее(x,y):  
    print('первое число равно', x)print('второе число равно', y)a = (x + y) /2.0  
    print('среднее значение равно', a)return a
```

Обсудим, что делают эти команды. Первые две строки, начинающиеся с символов #, Python игнорирует, но мы используем их в качестве комментария для потенциальных читателей кода. Следующая за ними конструкция `def среднее (x, y):` сообщает Python, что мы собираемся определить новую повторно используемую функцию. Здесь `def` (от англ. *define* – определить) – ключевое слово; `среднее` – имя, которое мы даем функции. Его можно выбирать произвольно, но лучше всего использовать *описательные имена*, которые напоят нам о том, для чего данная функция фактически предназначена. Конструкция в круглых скобках `(x,y)` сообщает.

Python, что функция принимает два входных значения, которые в пределах последующего определения функции обозначаются как `x` и `y`. В некоторых языках программирования требуется, чтобы вы указывали, объекты какого типа представляют переменные, но Python этого не делает, и впоследствии может лишь напомнить об использовании переменной не по назначению, например, когда вы пытаетесь использовать слово, как будто оно является числом и т.д.

Теперь, когда мы объявили Python о своем намерении определить функцию, мы должны сообщить, что именно делает эта функция. *Определение функции вводится с отступом*, что отражено в приведенном выше коде. В некоторых языках для того, чтобы было понятно, к каким частям программы относятся те или иные фрагменты кода, используется множество всевозможных скобок, но создатели Python осознавали, что обилие скобок затрудняет чтение кода, тогда как отступы позволяют с первого взгляда получить представление о его структуре. В отношении целесообразности такого подхода мнения разделились, поскольку люди иногда забывают о важности отступов.

Понять смысл кода в определении функции `среднее (x, y)` будет совсем несложно, поскольку в нем используется все то, с чем мы уже сталкивались. Этот код выводит числа, передаваемые функции при ее вызове. Для вычисления среднего значения выводить аргументы вовсе не обязательно, но мы делаем это для того, чтобы было совершенно понятно, что происходит внутри функции. В следующей инструкции вычисляется величина  $(x + y) / 2.0$ , и полученное значение присваивается переменной `a`. Мы выводим среднее значение исключительно для контроля того, что происходит в коде. Последняя инструкция – `return a`. Она завершает определение функции и сообщает Python, что именно должна вернуть функция в качестве результата, подобно машинам, которые мы ранее обсуждали.

Запустив этот код, вы увидите, что ничего не произошло. Никакие числа не выведены. Дело в том, что мы всего лишь определили функцию, но пока что не вызвали ее. На самом деле Python зарегистрировал функцию и будет держать ее наготове до тех пор, пока мы не захотим ее использовать.

```
In [7]: # функция, которая принимает два числа в качестве входных данных
# и выводит их среднее значение
def среднее(x,y):
    print('первое число равно', x)
    print('второе число равно', y)
    a = (x + y) / 2.0
    print('среднее значение равно', a)
    return a
```

Введите в следующей ячейке *среднее* (2, 4), чтобы активизировать функцию для значений 2 и 4. Кстати, в мире компьютерного программирования это называется **вызовом функции**. В соответствии с нашими ожиданиями данная функция выведет два входных значения и их вычисленное среднее. Кроме того, вы увидите ответ, выведенный отдельно, поскольку вызовы функций в интерактивных сеансах Python сопровождаются последующим выводом возвращаемого ими значения. Ниже показано определение функции, а также результаты ее вызова с помощью инструкции

*среднее* (2, 4) и инструкции *среднее* (200, 301) с большими значениями.

```
In [8]: среднее (2, 4)

первое число равно 2
второе число равно 4
среднее значение равно 3.0
```

Out[8]: 3.0

Поэкспериментируйте самостоятельно, вызывая функцию с различными входными значениями.

```
In [9]: среднее (200, 301)

первое число равно 200
второе число равно 301
среднее значение равно 250.5
```

Out[9]: 250.5

Возможно, вы обратили внимание на то, что в коде функции *среднее* двух значений находится путем деления не на 2, а на 2,0. Почему мы так поступили? Это особенность Python. Если бы мы делили на 2, то результат был бы округлен до ближайшего целого в меньшую сторону, поскольку Python рассматривал бы 2 просто как целое число. Это не изменило бы результат вызова *среднее* (2, 4), поскольку  $6/2$  равно 3, т.е. целому числу. Однако в случае вызова *среднее* (200, 301) среднее значение, равное  $501/2$ , т.е. 250,5, было бы округлено до значения 250. Деление же на 2,0 сообщает Python, что в наши намерения действительно входит работа с числами, которые могут иметь дробную часть, и мы не хотим, чтобы они округлялись.

Итак, мы определили повторно используемую функцию – один из наиболее важных и мощных элементов как математики, так и компьютерного программирования.

Мы будем применять повторно используемые функции при написании кода собственной нейронной сети. Например, имеет смысл создать повторно используемую функцию, которая реализовала бы вычисления с помощью сигмоиды, чтобы ее можно было вызывать много раз. Об этом в следующих работах.

## 2. Массивы

Массивы – это не более чем таблицы значений. Как и в случае таблиц, вы можете ссылаться на конкретные ячейки по номерам строк и столбцов.

Наверное, вам известно, что именно таким способом можно ссылаться на ячейки в электронных таблицах (например, B1 или C5) и производить над ними вычисления (например, C3+D7).

Когда дело дойдет до написания кода нейронной сети, мы используем массивы для представления матриц входных сигналов, весовых коэффициентов и выходных сигналов. Но не только этих, а также матриц, представляющих сигналы внутри нейронной сети и их распространение в прямом направлении, и матриц, представляющих обратное распространение ошибок. Поэтому давайте познакомимся с массивами. Введите и выполните следующий код:

```
import numpy
```

Что делает этот код? Команда `import` сообщает Python о необходимости привлечения дополнительных вычислительных ресурсов из другого источника для расширения круга имеющихся инструментов. В некоторых случаях эти дополнительные инструменты являются частью Python, но они не находятся в состоянии готовности к немедленному использованию, чтобы без надобности не отягощать Python. Чаще всего расширения не относятся к основному инструментарию Python, но создаются сторонними разработчиками в качестве вспомогательных ресурсов, доступных для всеобщего использования. В данном случае мы импортируем дополнительный набор инструментов, объединенных в единый модуль под названием **numpy**. Пакет `numpy` очень популярен и предоставляет ряд полезных средств, включая массивы и операции над ними.

Введите в следующей ячейке приведенный ниже код.

```
a= numpy.zeros( [3,2] )print(a)
```

В этом коде импортированный модуль `numpy` используется для создания массива размерностью 3x2, во всех ячейках которого содержатся нулевые значения. Мы сохраняем весь массив в переменной `a`, после чего выводим ее на экран. Результат подтверждает, что массив действительно состоит из нулей, хранящихся в виде таблицы с тремя строками и двумя столбцами.

```
In [10]: import numpy

In [11]: a= numpy.zeros( [3,2] )
          print(a)

[[0. 0.]
 [0. 0.]
 [0. 0.]
```

А теперь модифицируем содержимое этого массива и заменим некоторые из его нулей другими значениями. В приведенном ниже коде показано, как сослаться на конкретные ячейки для того, чтобы заменить хранящиеся в них значения новыми. Это напоминает обращение к нужным ячейкам электронной таблицы.

```
a [0,0] = 1
a [0,1] =2
a[1,0] = 9
a [2,1] = 12
print(a)
```

Первая строка обновляет ячейку, находящуюся в строке 0 и столбце 0, заменяя все, что в ней до этого хранилось, значением 1. В остальных строках другие ячейки аналогичным образом обновляются, а последняя строка выводит массив на экран с помощью инструкции **`print(a)`**. На приведенной ниже иллюстрации показано, что собой представляет массив после всех изменений.

```
In [16]: a [0,0] = 1
          a [0,1] =2
          a[1,0] = 9
          a [2,1] = 12
          print(a)

[[ 1.  2.]
 [ 9.  0.]
 [ 0. 12.]
```

Теперь, когда известно, как присваивать значения элементам массива, рассмотрим, каким образом можно узнать значение отдельного элемента, не выводя на экран весь массив. Чтобы сослаться на содержимое ячейки массива, которое мы хотим вывести на экран или присвоить другой переменной, достаточно воспользоваться выражением наподобие `a [ 0 ,1 ]` или `a [ 1 ,0 ]`. Именно это демонстрирует приведенный ниже фрагмент кода.

```
print(a[0,1])v = a [1,0]
print(v)
```

Запустив этот пример, вы увидите, что первая инструкция `print` выводит значение 2,0, т.е. содержимое ячейки `[0,1]`. Следующая инструкция присваивает значение элемента массива `a [1,0]` переменной `v` и выводит значение этой переменной. Как и ожидалось, выводится значение 9,0.

```
In [17]: print(a[0,1])
         v = a [1,0]
         print(v)
```

```
2.0
9.0
```

Нумерация столбцов и строк начинается с 0, а не с 1. Верхний левый элемент обозначается как `[0,0]`, а не как `[1,1]`. Отсюда следует, что правому нижнему элементу соответствует обозначение `[2,1]`, а не `[3,2]`. Если мы попытаемся, к примеру, обратиться к элементу массива `a[0,2]`, то получим сообщение об ошибке.

```
In [18]: #попытка обращения к несуществующему элементу массива
         a[0,2]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-18-f5ac02a5343a> in <module>
      1 #попытка обращения к несуществующему элементу массива
----> 2 a[0,2]

IndexError: index 2 is out of bounds for axis 1 with size 2
```

Массивы, или матрицы, пригодятся нам для нейронных сетей, поскольку позволят упростить инструкции при выполнении интенсивных вычислений, связанных с распространением сигналов и обратным распространением ошибок в сети.

### **3. Графическое представление массивов**

Как и в случае больших числовых таблиц и списков, понять смысл чисел, содержащихся в элементах массива большой размерности, довольно трудно. В то же время их визуализация позволяет быстро получить общее представление о том, что они означают. Одним из способов графического отображения двухмерных числовых массивов является их представление в виде двухмерных поверхностей, окраска которых в каждой точке зависит от значения соответствующего элемента массива. Способ установления соответствия между цветом поверхности и значением элемента массива вы выбираете по своему усмотрению. Например, можно преобразовывать значения в цвет, используя какую-либо цветовую шкалу, или закрасить всю поверхность белым цветом за исключением черных точек, которым соответствуют значения, превышающие определенный порог.

Давайте представим в графическом виде созданный ранее небольшой массив размерностью 3 x 2 .

Но сначала мы должны расширить возможности Python для работы с графикой. Для этого необходимо импортировать дополнительный код Python, написанный другими людьми. Это все равно, что взять у товарища книгу рецептов, поставить ее на свою книжную полку и пользоваться ею для приготовления блюд, которые раньше не умели готовить.

Ниже приведена инструкция, с помощью которой мы импортируем нужный нам пакет для работы с графикой:

```
import matplotlib.pyplot
```

Здесь `matplotlib.pyplot` – это имя новой «книги рецептов», которую мы на время одалживаем. Во всех подобных случаях имя модуля или библиотеки, предоставляющей в ваше распоряжение дополнительный код, указывается после ключевого слова `import`. В процессе работы с Python часто приходится импортировать дополнительные средства, облегчающие жизнь программиста за счет повторного использования полезного кода, предложенного сторонними разработчиками. Возможно, и вы разработаете когда-нибудь код, которым поделитесь с коллегами.

Кроме того, мы должны дополнительно сообщить IPython о том, что любую графику следует отображать в блокноте, а не в отдельном окне. Это делается с помощью следующей

директивы:

```
%matplotlib inline
```

Теперь мы полностью готовы к тому, чтобы представить массив в графическом виде.

Введите и выполните следующий код:

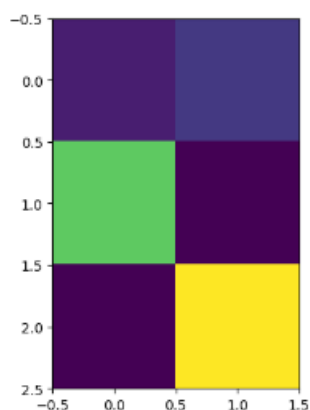
```
matplotlib.pyplot.imshow(a, interpolation="nearest")
```

```
[9]: import numpy
a= numpy.zeros( [3,2] )
a [0,0] = 1
a [0,1] =2
a[1,0] = 9
a [2,1] = 12

•[10]: import matplotlib.pyplot
%matplotlib inline

[11]: matplotlib.pyplot.imshow(a, interpolation="nearest")

[11]: <matplotlib.image.AxesImage at 0x36e71c0>
```



Нам удалось представить содержимое массива в виде цветной диаграммы. Вы видите, что ячейки, содержащие одинаковые значения, закрашены одинаковыми цветами. Позже мы используем ту же функцию `imshow()` для визуализации значений, которые будем получать из нашей нейронной сети.

В состав пакета IPython входит богатый набор инструментов, предназначенных для визуализации данных. Функция `imshow()` предлагает множество опций графического представления данных, таких как использование различных цветовых палитр.

#### 4.Объекты

Далее следует познакомиться с еще одним фундаментальным понятием, которое используется в Python, – понятием объекта. Объекты в чем-то схожи с повторно используемыми функциями, поскольку, однажды определив их, вы сможете впоследствии обращаться к ним множество раз. Но по сравнению с функциями объекты способны на гораздо большее.

Запишите следующий код: # класс объектов Dog (собака) `class Dog:`

```
# собаки могут лаять def bark(self):
```

```
print("Гав!") pass
```

```
pass
```

Начнем с того, с чем мы уже знакомы. Прежде всего, код включает функцию `bark()`. Как нетрудно заметить, при вызове данной функции она выведет слово “Гав!” А теперь рассмотрим код в целом. Вы видите ключевое слово `class`, за которым следуют имя `Dog` (собака) и структура, напоминающая функцию. Вы сразу же можете провести параллели между

этой структурой и определением функции, которое также снабжается именем. Отличаются же они тем, что для определения функций используется ключевое слово *def*, тогда как для определения объектов используется ключевое слово *class*.

Прежде чем углубиться в обсуждение того, какое отношение эта структура, называемая **классом**, имеет к объектам, вновь к простому, нореальному коду, который оживляет эти понятия.

```
sizzles = Dog()sizzles.bark()
```

В первой строке создается переменная *sizzles*, источником которой является вызов функции. *Dog ()* – это особая функция, которая создает экземпляр класса *Dog*. Таким образом можно создавать различные сущности из определений классов. Эти сущности называются объектами. В данном случае мы создали из определения класса *Dog* объект *sizzles* и можем считать, что этот объект является собакой.

В следующей строке для объекта *sizzles* вызывается функция *bark()*. Функция *bark ()* вызывается так, как если бы она была частью объекта *sizzles*. Это возможно, потому что данную функцию имеют все объекты, созданные на основе класса *Dog*, ведь именно в нем она и определена.

Опишем все это простыми словами. Мы создали переменную *sizzles*, разновидность класса *Dog*. Переменная *sizzles* – это объект, созданный по шаблону класса *Dog*. Объекты – это экземпляры класса.

Следующий пример показывает, что мы к этому времени успели сделать, и подтверждает, что функция *sizzles.bark()* действительно выводит слово “Гав!”

```
[13]: # класс объектов Dog (собака)
class Dog:
    # собаки могут лаять
    def bark(self):
        print("Гав!")
        pass
        pass
```

```
[14]: sizzles = Dog()
```

```
[15]: sizzles.bark()

Гав!
```

Возможно, вы обратили внимание на элемент *self* в определении функции – *bark (self)*. Он здесь для того, чтобы указывать Python, к какому объекту приписывается функция при ее создании.

Рассмотрим примеры более полезного использования объектов и классов. Наберите на следующий код:

```
sizzles = Dog ()mutley = Dog()
```

```
sizzles.bark()mutley.bark()
```

```
[16]: sizzles = Dog ()
mutley = Dog()

sizzles.bark()
mutley.bark()
```

```
Гав!
Гав!
```

Мы создали два объекта: *sizzles* и *mutley*. **Важно понимать, что оба они были созданы на основе одного и того же определения класса *Dog*. Сначала мы определяем, как объекты должны выглядеть и как должны себя вести, а затем создаем их реальные экземпляры.**

В этом и состоит суть различия между классами и объектами: первые представляют собой описания объектов, а вторые – реальные экземпляры классов, созданные в соответствии с этими описаниями. *Класс* – это рецепт приготовления пирога, а *объект* – сам пирог, изготовленный по данному рецепту. Процесс создания объектов на основе описания класса можно пояснить с помощью следующей наглядной иллюстрации.

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Искусственный интеллект и естественный язык.
2. Естественный язык и естественность общения человека с системой ИИ.
3. Понимание выражений естественного языка.
4. Представление лингвистических знаний и методы анализа и синтеза текста.
5. СИИ и прикладные системы обработки текста.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

### **Проект нейронной сети на Python. Часть 1.**

#### **1. Создание класса нейронной сети.**

Класс нейронной сети должен содержать, по крайней мере, три функции:

- *инициализация* – задание количества входных, скрытых и выходных узлов;
- *тренировка* – уточнение весовых коэффициентов в процессе обработки предоставленных для обучения сети тренировочных примеров;
- *опрос* – получение значений сигналов с выходных узлов после предоставления значений входящих сигналов.

Начальный код может иметь примерно следующий вид.

```
# определение класса нейронной сети
class neuralNetwork:
    # инициализировать нейронную сеть
    def __init__():
        pass
    # тренировка нейронной сети
    def train():
        pass
    # опрос нейронной сети
    def query():
        pass
```

#### **2. Инициализация сети.**

Начнем с инициализации. Необходимо задать количество узлов входного, скрытого и выходного слоев. Эти данные определяют конфигурацию и размер нейронной сети. Вместо того чтобы жестко задавать их в коде, мы предусмотрим установку соответствующих значений в виде параметров во время создания объекта нейронной сети. Благодаря этому можно будет без труда создавать новые нейронные сети различного размера.

В основе нашего решения лежит одно важное соображение. Хорошие программисты при каждом удобном случае стараются писать *обобщенный код*, не зависящий от конкретных числовых значений. Это хорошая привычка, поскольку она вынуждает нас более глубоко продумывать решения, расширяющие применимость программы. Следуя ей, мы сможем использовать наши программы в самых разных сценариях. В данном случае это означает, что мы будем пытаться разрабатывать код для нейронной сети, поддерживающий как можно больше открытых опций и использующий как можно меньше предположений, чтобы его можно было легко применять в различных ситуациях. Мы хотим, чтобы один и тот же класс мог создавать как небольшие нейронные сети, так и очень большие, требуя лишь задания желаемых размеров сети в качестве параметров.

Кроме того, нам нельзя забывать о коэффициенте обучения. Этот параметр также можно устанавливать при создании новой нейронной сети. Посмотрите, как может выглядеть функция инициализации `__init__()` в подобном случае и **добавьте нужный код**.

```

# определение класса нейронной сети
class neuralNetwork:

    # инициализировать нейронную сеть
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # задать количество узлов во входном, скрытом и выходном слое
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # коэффициент обучения
        self.lr = learningrate
        pass

    # тренировка нейронной сети
    def train():
        pass

    # опрос нейронной сети
    def query():
        pass

```

Добавим этот код в наше определение класса нейронной сети и создадим объект небольшой сети с тремя узлами в каждом слое и коэффициентом обучения, равным 0,3.

Данный код позволяет получить объект сети, но такой объект пока что не будет особенно

```

[3]: # определение класса нейронной сети
class neuralNetwork:

    # инициализировать нейронную сеть
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # задать количество узлов во входном, скрытом и выходном слое
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # коэффициент обучения
        self.lr = learningrate
        pass

    # тренировка нейронной сети
    def train():
        pass

    # опрос нейронной сети
    def query():
        pass

```

```

[4]: input_nodes = 3
hidden_nodes = 3
output_nodes = 3

# коэффициент обучения равен 0,3
learning_rate = 0.3

# создать экземпляр нейронной сети
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes,learning_rate)

```

полезным, потому что не содержит ни одной функции, способной выполнять полезную работу.

Это нормальная практика – начинать с малого и постепенно наращивать код, попутно находя и устраняя ошибки.

Что дальше? Мы сообщили объекту нейронной сети, сколько узлов разных типов нам необходимо иметь, но для фактического создания узлов пока что ничего не сделали.

### **3. Весовые коэффициенты – основная часть сети.**

Нашим следующим шагом будет создание сети, состоящей из узлов и связей. Наиболее важная часть этой сети – **весовые коэффициенты связей (веса)**. Они используются для расчета распространения сигналов в прямом

направлении, а также обратного распространения ошибок, и именно весовые коэффициенты уточняются в попытке улучшить характеристики сети.

Удобна компактная запись весов в виде матриц. Следовательно, мы можем создать следующие матрицы:

- матрицу весов для связей между входным и скрытым слоями,  
**W** входной\_скрытый, размерностью **hidden\_nodes x input\_nodes**;
- другую матрицу для связей между скрытым и выходным слоями,  
**W** скрытый\_выходной, размерностью **output\_nodes x hidden\_nodes**.

При необходимости вернитесь к теоретическим материалам лабораторных работ, чтобы понять, почему первая матрица имеет размерность **hidden\_nodes x input\_nodes**, а не **input\_nodes x hidden\_nodes**.

Вспомним, что начальные значения весовых коэффициентов должны быть небольшими и выбираться случайным образом. Следующая функция из пакета *numpy* генерирует массив случайных чисел в диапазоне от 0 до 1, где размерность массива равна **rows x columns**:

```
numpy.random.rand(rows, columns)
```

Все хорошие программисты ищут в Интернете онлайн-документацию по функциям Python и находят полезные функции. При желании выполните такой поиск для функции `numpy.random.rand()`, если хотите узнать о ней больше.

Если мы собираемся использовать расширения пакета *numpy*, мы должны импортировать эту библиотеку в самом начале кода. Прежде всего удостоверимся, что нужная нам функция работает. В приведенном ниже примере используется матрица размерностью 3x3. Как видите, матрица заполнилась случайными значениями в диапазоне от 0 до 1.

Данный подход можно улучшить, поскольку весовые коэффициенты могут иметь

```
[5]: import numpy
```

```
[6]: numpy.random.rand(3,3)
```

```
[6]: array([[0.96043689, 0.17240669, 0.04986505],
          [0.29109845, 0.73728517, 0.24248404],
          [0.68053887, 0.05742121, 0.43125263]])
```

не только положительные, но и отрицательные значения и изменяться в пределах от -1,0 до +1,0. Для простоты мы вычтем 0,5 из этих граничных значений, перейдя к диапазону значений от -0,5 до +0,5. Ниже представлены результаты применения этого подхода, которые демонстрируют, что некоторые весовые коэффициенты теперь имеют отрицательные значения.

```
[7]: numpy.random.rand(3,3) - 0.5
```

```
[7]: array([[ 0.21210893, -0.20506655,  0.29219392],
          [-0.38341736,  0.31064623, -0.33060099],
          [-0.00113976, -0.01018176,  0.39300294]])
```

Мы готовы создать матрицу начальных весов в нашей программе на Python. Эти весовые коэффициенты составляют неотъемлемую часть нейронной сети и служат ее характеристиками на протяжении всей ее жизни, а не временным набором данных, которые исчезают сразу же после того, как функция отработала. Это означает, что они должны быть частью процесса инициализации и быть доступными для других методов, таких как функции тренировки и опроса сети.

Ниже приведен код, включая комментарии, который создает две матрицы весовых коэффициентов, используя значения переменных `self.inodes`, `self.hnodes` и `self.onodes` для задания соответствующих размеров каждой из них.

```
1]: # Матрицы весовых коэффициентов связей wih (между входным и скрытым
# слоями) и who (между скрытым и выходным слоями).
# Весовые коэффициенты связей между узлом i и узлом j следующего слоя
# обозначены как w_i_j:
# w11 w21
# w12 w22 и т.д.
self.wih = (numpy.random.rand(self.hnodes, self.inodes) - 0.5)
self.who = (numpy.random.rand(self.onodes, self.hnodes) - 0.5)
```

#### **4. Улучшенный вариант инициализации весовых коэффициентов.**

Описанное в этом разделе *обновление кода* не является обязательным, поскольку это всего лишь простое, но популярное усовершенствование процесса инициализации весовых коэффициентов.

В теоретических материалах лабораторных работ мы рассматривали различные способы подготовки данных и инициализации коэффициентов. Можно применить и другой, несколько усовершенствованный подход к созданию случайных начальных значений весов. Для этого весовые коэффициенты выбираются из нормального распределения с центром в нуле и со стандартным отклонением, величина которого обратно пропорциональна корню квадратному из количества входящих связей на узел.

Это легко делается с помощью библиотеки *numpy*. Функция *numpy.random.normal()* описана по такому адресу: <https://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.random.normal.html>. Она поможет нам с извлечением выборки из нормального распределения. Ее параметрами являются центр распределения, стандартное отклонение и размер массива *numpy*, если нам нужна матрица случайных чисел, а не одиночное число.

Обновленный код инициализации весовых коэффициентов будет выглядеть так.

Центр нормального распределения установлен здесь в 0,0. Стандартное отклонение

```
self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))
self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, self.hnodes))
```

вычисляется по количеству узлов в следующем слое с помощью функции *pow(self.hnodes, - 0.5)*, которая просто возводит количество узлов в степень -0,5 . Последний параметр определяет конфигурацию массива *numpy*.

### **ЛР №7. Решение задач и искусственный интеллект**

**Задание 1:** подготовить конспекты и подготовиться к собеседованию по следующим вопросам.

1. Представление задач в пространстве состояний.
2. Стратегии поиска решения: методы полного перебора
3. Редукция задач.
4. Планирование действий. Роботы и искусственный интеллект.
5. Интеллектуальные агенты.

**Задание 2.** Проанализировать итоги собеседования и выполнить задание для самостоятельной работы.

#### **1. Опрос сети**

Далее зададим функцию *query()*, которая обеспечивает опрос сети – получение значений сигналов с выходных узлов после предоставления значений входящих сигналов. Т.е. функция *query()* принимает в качестве аргумента входные данные нейронной сети и возвращает ее выходные данные. Для этого нам нужно передать сигналы от узлов входного слоя через скрытый слой к узлам выходного слоя для получения выходных данных. При этом, как вы помните, по мере распространения сигналов мы должны сглаживать их, используя весовые коэффициенты связей между соответствующими узлами, а также применять сигмоиду для уменьшения выходных сигналов узлов.

В случае большого количества узлов написание для каждого из них кода на Python, осуществляющего сглаживание весовых коэффициентов, суммирование сигналов и применение к ним сигмолды, превратилось бы в сплошной кошмар.

К счастью, нам не нужно писать детальный код для каждого узла, поскольку мы уже знаем, как записать подобные инструкции в простой и компактной матричной

форме. Ниже показано, как можно получить входящие сигналы для узлов скрытого слоя путем сочетания матрицы весовых коэффициентов связей между входным и скрытым слоями с матрицей входных сигналов:

$$X_{\text{скрытый}} = W_{\text{входной\_скрытый}} * I$$

Компьютерные языки, в том числе Python, распознают матрицы и эффективно выполняют все расчеты, поскольку им известно об однотипности всех стоящих за этим вычислений.

Код на языке Python для этой операции достаточно простой. Ниже представлена инструкция, которая показывает, как применить функцию скалярного произведения библиотеки *numpy* к матрицам весов и входных сигналов:

Эта короткая строка кода Python выполняет всю работу по объединению всех

```
] hidden_inputs = numpy.dot(self.wih, inputs)
```

входных сигналов с соответствующими весами для получения матрицы сглаженных комбинированных сигналов в каждом узле скрытого слоя. Более того, нам не придется ее переписывать, если в следующий раз мы решим использовать входной или скрытый слой с другим количеством узлов. Этот код все равно будет работать!

Для получения выходных сигналов скрытого слоя мы просто применяем к каждому из них сигмоиду:

$$O_{\text{скрытый}} = \text{сигмоида}(X_{\text{скрытый}})$$

Это не должно вызвать никаких затруднений, особенно если сигмоида уже определена в какой-нибудь библиотеке Python.

Библиотека *scipy* в Python содержит набор специальных функций, в том числе сигмоиду, которая называется в данной библиотеке *expit()*. Библиотека *scipy* импортируется точно так же, как и библиотека *numpy*.

```
] # библиотека scipy.special содержит сигмоиду expit()
import scipy.special
```

Поскольку в будущем мы можем захотеть поэкспериментировать с функцией активации, настроив ее параметры или полностью заменив другой функцией, лучше определить ее один раз в объекте нейронной сети во время его инициализации. После этого мы сможем неоднократно ссылаться на нее, точно так же, как на функцию *query()*. Такая организация программы означает, что в случае внесения изменений нам придется сделать это только в одном месте, а не везде в коде, где используется функция активации. Ниже приведен код, определяющий функцию активации, который мы используем в разделе инициализации нейронной сети.

```
# использование сигмоиды в качестве функции активации
self.activation_function = lambda x: scipy.special.expit(x)
```

Что делает этот код? Он создает функцию наподобие любой другой, только с использованием более короткого способа записи, называемого *лямбда-выражением*. Вместо привычного определения функции в форме *def имя()* мы использовали слово *lambda*, которое позволяет создавать функции быстрым и удобным способом, что называется, «на лету». В данном случае функция принимает аргумент *x* и возвращает *scipy.special.expit()*, а это есть не что иное, как сигмоида. Функции, создаваемые с помощью лямбда-выражений, являются безымянными или, как предпочитают говорить опытные программисты, анонимными, но данной функции мы присвоили имя *self.activation\_function()*. Это означает, что всякий раз, когда потребуется использовать функцию активации, ее нужно будет вызвать как *self.activation\_function()*.

Итак, возвращаясь к нашей задаче, мы применим функцию активации к сглаженным комбинированным входящим сигналам, поступающим на скрытые узлы. Соответствующий код совсем не сложен.

```
# рассчитать исходящие сигналы для скрытого слоя  
hidden_outputs = self.activation_function(hidden_inputs)
```

Таким образом, сигналы, исходящие из скрытого слоя, описываются матрицей *hidden\_outputs*.

Мы прошли промежуточный скрытый слой, а как быть с последним, выходным слоем? В действительности распространение сигнала от скрытого слоя до выходного ничем принципиально не отличается от предыдущего случая, поэтому способ расчета остается тем же, а значит, и код будет аналогичен предыдущему.

Ниже приведен итоговый фрагмент кода, объединяющий расчеты сигналов скрытого и выходного слоев.

```
# рассчитать входящие сигналы для скрытого слоя  
hidden_inputs = numpy.dot(self.wih, inputs)  
# рассчитать исходящие сигналы для скрытого слоя  
hidden_outputs = self.activation_function(hidden_inputs)  
# рассчитать входящие сигналы для выходного слоя  
final_inputs = numpy.dot(self.who, hidden_outputs)  
# рассчитать исходящие сигналы для выходного слоя  
final_outputs = self.activation_function(final_inputs)
```

Если отбросить комментарии, здесь всего четыре строки кода, выделенные полужирным шрифтом, которые выполняют все необходимые расчеты: две – для скрытого слоя и две – для выходного слоя.

## **2. Текущее состояние кода**

Посмотрим, как выглядит в целом код, который мы к этому времени создали.

```

7]: # определение класса нейронной сети
class neuralNetwork:

    # инициализировать нейронную сеть
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # задать количество узлов во входном, скрытом и выходном слое
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        # Матрицы весовых коэффициентов связей wih и who.
        # Весовые коэффициенты связей между узлом i и узлом j
        # следующего слоя обозначены как w_i_j:
        # w11 w21
        # w12 w22 и т.д.
        self.wih = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.onodes, -0.5), (self.onodes, self.hnodes))

        # коэффициент обучения
        self.lr = learningrate
        # использование сигмной функции в качестве функции активации
        self.activation_function = lambda x: scipy.special.expit(x)

    pass

    # тренировка нейронной сети
    def train():
        pass

    # опрос нейронной сети
    def query(self, inputs_list):
        # преобразовать список входных значений
        # в двумерный массив
        inputs = numpy.array(inputs_list, ndmin=2).T
        # рассчитать входящие сигналы для скрытого слоя
        hidden_inputs = numpy.dot(self.wih, inputs)
        # рассчитать исходящие сигналы для скрытого слоя
        hidden_outputs = self.activation_function(hidden_inputs)
        # рассчитать входящие сигналы для выходного слоя
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # рассчитать исходящие сигналы для выходного слоя
        final_outputs = self.activation_function(final_inputs)
        return final_outputs

```

Но это только определение класса, перед которым в первой ячейке блокнота Python следует поместить код, импортирующий модули *numpy* и *scipy.special*.

Функции *query()* в качестве входных данных потребуются только входные сигналы *input\_list*. Ни в каких других входных данных она не нуждается.

```

In [8]: import numpy
        # библиотека scipy.special с сигмной функцией expit()
        import scipy.special

```

Мы достигли значительного прогресса и теперь можем вернуться к недостающему фрагменту – функции *train()*. Вспомните, что тренировка включает две фазы: *первая* – это расчет выходного сигнала, что и делает функция *query()*, а *вторая* – обратное распространение ошибок, информирующее вас о том, каковы должны быть поправки к весовым коэффициентам. Прежде чем переходить к написанию кода функции *train()*, осуществляющей тренировку сети на примерах, протестируем, как работает уже имеющийся код. Для этого создадим небольшую сеть и предоставим ей некоторые входные данные. Очевидно, что никакого реального смысла результаты содержать не будут, но нам важно лишь проверить работоспособность всех созданных функций.

В следующем примере создается небольшая сеть с входным, скрытым и выходным слоями, содержащими по три узла, которая опрашивается с использованием случайно выбранных входных сигналов (1.0, 0.5 и -1.5).

```

.8]: input_nodes = 3
      hidden_nodes = 3
      output_nodes = 3

      # коэффициент обучения равен 0,3
      learning_rate = 0.3

      # создать экземпляр нейронной сети
      n = neuralNetwork(input_nodes, hidden_nodes, output_nodes, learning_rate)

.9]: n.query([1.0, 0.5, -1.5])

.9]: array([[0.63875786],
           [0.65613726],
           [0.43678544]])

```

Нетрудно заметить, что при создании объекта нейронной сети необходимо задавать значение коэффициента обучения, даже если он не используется. Это объясняется тем, что определение класса нейронной сети

включает функцию инициализации `init` `__init__()`, которая требует

предоставления данного аргумента. Если его не указать, программа не сможет быть выполнена, и отобразится сообщение об ошибке.

Вы также могли заметить, что входные данные передаются в виде списка, который в Python обозначается квадратными скобками. Вывод также представлен в виде числового списка. Эти значения не имеют никакого реального смысла, поскольку мы не тренировали сеть, но тот факт, что во время выполнения программы не возникли ошибки, должен нас радовать.

### 3. Тренировка сети

Приступим к решению несколько более сложной задачи тренировки сети. Ее можно разделить на две части.

- Первая часть — расчет выходных сигналов для заданного тренировочного примера. Это ничем не отличается от того, что мы уже можем делать с помощью функции `query()`.
- Вторая часть — сравнение рассчитанных выходных сигналов с желаемым ответом и обновление весовых коэффициентов связей между узлами на основе найденных различий.

Сначала запишем готовую первую часть.

```

# тренировка нейронной сети
def train(self, inputs_list, targets_list):
    # преобразование списка входных значений
    # в двумерный массив
    inputs = numpy.array(inputs_list, ndmin=2).T
    targets = numpy.array(targets_list, ndmin=2).T

    # рассчитать входящие сигналы для скрытого слоя
    hidden_inputs = numpy.dot(self.wih, inputs)
    # рассчитать исходящие сигналы для скрытого слоя
    hidden_outputs = self.activation_function(hidden_inputs)
    # рассчитать входящие сигналы для выходного слоя
    final_inputs = numpy.dot(self.who, hidden_outputs)
    # рассчитать исходящие сигналы для выходного слоя
    final_outputs = self.activation_function(final_inputs)

```

Этот код почти совпадает с кодом функции `query()`, поскольку процесс передачи сигнала от входного слоя к выходному остается одним и тем же.

Единственным отличием является введение дополнительного параметра `targets_list`, передаваемого при вызове функции, поскольку невозможно тренировать сеть без предоставления ей тренировочных примеров, которые включают желаемые или целевые значения:

```

def train(self, inputs_list, targets_list):

```

Список `targets_list` преобразуется в массив точно так же, как список `input_list`:

```
targets = numpy.array(targets_list, ndmin=2).T
```

Теперь мы очень близки к решению основной задачи тренировки сети – уточнению весов на основе расхождения между расчетными и целевыми значениями.

Будем решать эту задачу поэтапно.

Прежде всего, мы должны вычислить ошибку, являющуюся разностью между желаемым целевым выходным значением, предоставленным тренировочным примером, и фактическим выходным значением.

Она представляет собой разность между матрицами (`targets - final_outputs`), рассчитываемую поэлементно. Соответствующий код выглядит очень просто, что еще раз подтверждает мощь и красоту матричного подхода.

```
# ошибки выходного слоя =  
# (целевое значение - фактическое значение)  
output_errors = targets - final_outputs
```

Далее мы должны рассчитать обратное распространение ошибок для узлов скрытого слоя. Вспомните, как мы распределяли ошибки между узлами пропорционально весовым коэффициентам связей, а затем рекомбинировали их на каждом узле скрытого слоя. Эти вычисления можно представить в следующей матричной форме:

Код, реализующий эту формулу, также прост в силу способности Python вычислять скалярные произведения матриц с помощью модуля `numpy`.

```
# ошибки скрытого слоя - это ошибки output_errors,  
# распределенные пропорционально весовым коэффициентам связей  
# и рекомбинированные на скрытых узлах  
hidden_errors = numpy.dot(self.who.T, output_errors)
```

Итак, мы получили то, что нам необходимо для уточнения весовых коэффициентов в каждом слое. Для весов связей между скрытым и выходным слоями мы используем переменную `output_errors`.

Для весов связей между входным и скрытым слоями мы используем только что рассчитанную переменную `hidden_errors`.

Это выражение легко транслируется в код на языке Python. Сначала запишем код для обновления весов связей между скрытым и выходным слоями.

```
# обновить весовые коэффициенты для связей между  
# скрытым и выходным слоями  
self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))
```

Это довольно длинная строка кода, но цветовое выделение поможет вам разобраться в том, как она связана с приведенным выше математическим выражением. Коэффициент обучения `self.lr` умножается на остальную часть выражения. Есть еще матричное умножение, выполняемое с помощью функции `numpy.dot()`, и два элемента, выделенных синим и красным цветами, которые отображают части, относящиеся к ошибке и сигмоидам из следующего слоя, а также транспонированная матрица исходящих сигналов предыдущего слоя.

Операция `+=` означает увеличение переменной, указанной слева от знака равенства, на значение, указанное справа от него. Поэтому `x+=3` означает, что `x` увеличивается на 3. Это просто сокращенная запись инструкции `x=x+3`. Аналогичный способ записи допускается и для других арифметических операций. Например, `x/=3` означает деление `x` на 3. Код для уточнения весовых коэффициентов связей между входным и скрытым слоями будет очень похож на этот. Мы воспользуемся симметрией выражений и просто перепишем код, заменяя в нем имена переменных таким образом, чтобы они относились к предыдущим слоям. Ниже приведен суммарный код для двух наборов весовых коэффициентов, отдельные элементы которого выделены цветом таким образом, чтобы сходные и различающиеся участки кода можно было легко заметить.

```
# обновить весовые коэффициенты для связей между
# скрытым и выходным слоями
self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose(hidden_outputs))

# обновить весовые коэффициенты для связей между
# входным и скрытым слоями
self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))
```

## Список литературы

### Основная литература:

1. Пальмов С.В. Интеллектуальные системы и технологии Электронный ресурс: учебное пособие / С.В. Пальмов. - Самара: Поволжский государственный университет телекоммуникаций и информатики, 2017. - 195 с. - Книга находится в базовой версии ЭБС IPRbooks.

### Дополнительная литература:

1. Смолин, Д.В. Введение в искусственный интеллект : конспект лекций / Д.В. Смолин. - М. : ФИЗМАТЛИТ, 2007. - 264 с.

2. Сотник С.Л. Проектирование систем искусственного интеллекта Электронный ресурс: учебное пособие / С.Л. Сотник. - Проектирование систем искусственного интеллекта, 2021-01-23. - Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 228 с. - Книга находится в базовой версии ЭБС IPRbooks.

### Методическая литература:

1. Методические указания к практическим занятиям (электронный вариант)
2. Методические указания к самостоятельной работе (электронный вариант)

### Интернет-ресурсы:

1. Люгер Дж. Искусственный интеллект: стратегии и методы решения сложных проблем. 4-е издание. – М.: Вильямс, 2003. [DJVU] (<http://depositfiles.com/ru/files/vzsr83pq>)
2. Джексон П. Введение в экспертные системы. – М.: Вильямс, 2000. [DOC] (<http://www.twirpx.com/file/117162/>)
3. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. – С-Пб.: Питер, 2000. [DJVU] (<http://www.twirpx.com/file/13533/>)
4. Мальковский М.Г. Конспект лекций по курсу «Искусственный интеллект» [HTML] (<http://al.cs.msu.su/classes/ai>).

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ**  
**Федеральное государственное автономное образовательное учреждение  
высшего образования**  
**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Методические указания**

для обучающихся по организации и проведению самостоятельной работы  
по дисциплине «Системы искусственного интеллекта»  
для студентов направления 44.03.05 Педагогическое образование (с двумя  
профилями подготовки)  
направленность (профиль):  
География и биология

**Ставрополь 2026**

## **1. Общие положения**

Самостоятельная работа – планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента.

К основным видам самостоятельной работы студентов относятся:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- написание докладов;
- подготовка к семинарам, практическим и лабораторным работам, их оформление;
- составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);
- выполнение учебно-исследовательских работ, проектная деятельность;
- подготовка практических разработок и рекомендаций по решению проблемной ситуации;
- выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;
- компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;
- выполнение курсовых работ (проектов) в рамках дисциплин;
- выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

## **2. Цель и задачи самостоятельной работы**

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию

самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

### **3. Порядок выполнения самостоятельной работы магистрантом**

#### ***3.1. Методические рекомендации по работе с учебной литературой***

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные

сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют **четыре основные установки в чтении научного текста**:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

*Основные виды систематизированной записи прочитанного:*

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

*Методические рекомендации по составлению конспекта:*

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;

2. Выделите главное, составьте план;

3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;

4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

### **Вопросы для собеседования по самостоятельно изученной литературе**

1. Краткая история искусственного интеллекта (ИИ), машины и интеллект. Основные направления исследований в области ИИ.

2. Интеллектуальные системы общения: системы обработки текстов естественного языка, системы обучения с базами данных, диалоговые системы решения задач, системы речевого общения.

3. Разработка естественно-языковых интерфейсов и машинный перевод с одного языка на другой.

4. Интеллектуальные роботы.

5. Обучение и самообучение, искусственные нейронные сети.

6. Задачи распознавания образов.

7. Интеллектуальные игры и машинное творчество.

8. Представление задач и стратегии поиска их решения в пространстве состояний (поиск в глубину и ширину, слепой и эвристический поиск, поиск на игровых деревьях, минимаксный алгоритм, альфа-бета алгоритм и др.).

9. Представление знаний - центральная проблема ИИ. Процедурная и декларативная информация. Переход от обработки данных к оперированию со знаниями.

10. Отличительные особенности знаний от данных: внутренняя интерпретируемость, структурированность, связность, активность. Базы знаний. Нечеткие и неточные знания.

11. Открытость знаний в системах ИИ. Основные методы приобретения знаний. Инженерия знаний.

12. Основные модели представления знаний.

13. Формальные логические модели представления знаний. Проблема понимания естественного языка.

14. Сетевые модели представления знаний (семантические сети). Отображение множества информационных единиц во множество типов связей между ними. Отношения типа «абстрактное-конкретное» и «целое-часть». Иерархия наследования.

15. Фреймовые модели представления знаний. Понятия фрейма, терминального слота, протофрейма. Имя фрейма и имя слота, значение слота и тип данных слота. Фреймы-экземпляры. Механизм наследования.

16. Продукционная модель представления знаний. Системы продукций. База знаний (база фактов и правил).

17. Рабочая память. Механизм вывода: прямая и обратная цепочка рассуждений.

18. Достоинства и недостатки продукционной модели представления знаний.

19. Интегрированные модели представления знаний. Языки представления знаний.

20. Общая характеристика программных средств для разработки и реализации систем ИИ. Требования к программному обеспечению систем ИИ.

21. Инструментальные средства для создания систем ИИ.

22. Понятие экспертной системы (ЭС). Назначение, принципы построения и области применения ЭС. Организация знаний в ЭС.

23. Виды ЭС и типы решаемых ими задач. Инструментальные средства разработки ЭС. Оболочковые средства создания прототипов ЭС.

24. Гибридные логические и моделирующие ЭС, ЭС на базе нечеткой логики. Интеллектуальные информационные ЭС.

25. Структурная схема, основные компоненты, архитектура и режимы использования ЭС продукционного типа.

26. Основные этапы разработки ЭС. Жизненный цикл ЭС.

27. Языки программирования для решения задач ИИ.

### *3.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям*

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения

лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

#### *4.3. Методические рекомендации по самопроверке знаний*

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

#### *3.4. Методические рекомендации по написанию научных текстов (докладов, рефератов, эссе, научных статей и т.д.)*

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремиться структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т, п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору

(исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Реферат (доклад) – это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Реферат не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в реферате должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки реферата студентом.

Выполнение реферата начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы – изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания реферата. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста реферата предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки реферат сдается на кафедру для его оценивания руководителем.

#### *Требования к написанию реферата*

Написание 1 реферата является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема реферата может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Реферат должен быть написан научным языком.

Объем реферата должен составлять 20-25 стр.

#### *Структура реферата:*

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.
- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению.

Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.

- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса

- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.

- Приложение (при необходимости).

*Требования к оформлению:*

- текст с одной стороны листа;
- шрифт TimesNewRoman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- реферат должен быть представлен в сброшюрованном виде.

*Порядок защиты реферата:*

Защита реферата проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту реферата отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите реферата приветствуется использование мультимедиа-презентации.

*Оценка реферата*

Реферат оценивается по следующим критериям:

- соблюдение требований к его оформлению;
- необходимость и достаточность для раскрытия темы приведенной в тексте реферата информации;
- умение студента свободно излагать основные идеи, отраженные в реферате;
- способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

*Критерии оценки:*

*Оценка «отлично»* выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

*Оценка «хорошо»* выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

*Оценка «удовлетворительно»* выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

*Оценка «неудовлетворительно»* выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

**Темы докладов**

1. Моделирование биологических систем.
2. Автоматический компьютерный синтез речи по тексту. Методы синтеза речи.
3. Примеры систем обработки естественного языка.
4. Классификация инструментальных средств ЭС и организация знаний в ЭС.
5. Модели представления знаний: логическая, сетевая, фреймовая, продукционная.
6. Методы озвучивания речи.
7. История возникновения и современные направления исследований в области ИИ.
8. Классификация систем распознавания речи.
9. Распознавание символов. Шаблонные системы. Структурные системы. Признаковые системы.
10. Речевой вывод информации.
11. Машинный интеллект и робототехника.
12. Типы задач решаемые в ЭС.
13. Основные направления исследований в области искусственного интеллекта.
14. Предпосылки возникновения систем понимания естественного языка. Понимание в диалоге.
15. Распознавание рукописных текстов.

### *3.5. Методические рекомендации по выполнению исследовательских проектов*

Исследовательская проектная работа – это групповая работа, для выполнения которой необходим выбор и приложение научной методики к поставленной задаче, получение собственного теоретического или экспериментального материала, на основании которого необходимо провести анализ и сделать выводы об исследуемом явлении. Выполнение проекта – это всегда коллективная, творческая Лабораторная работа, предназначенная для получения определенного продукта или научно-технического результата. Такая работа подразумевает четкое, однозначное формирование поставленной задачи, определение сроков выполнения намеченного, определение требований к разрабатываемому объекту.

Выполнение 1 группового проекта является обязательным условием выполнения самостоятельной работы по любой дисциплине профессионального цикла. Тема проектного задания может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

#### *Требования по выполнению и оформлению проекта*

При выполнении проекта приветствуется работа в группе (2-3 человека). Проект – это исследовательская работа, в ходе которой студенты должны продемонстрировать владение навыками научного исследования, умения проводить анализ, обобщать информацию, делать выводы, предлагать свои решения проблемы, рассматриваемой в проекте.

При подготовке материалов проекта студенты должны продемонстрировать владение современными методами компьютерной обработки данных.

#### *Критерии оценки работы участника проекта.*

Для каждого из участников проекта оцениваются:

- профессиональные теоретические знания в соответствующей области;
- умение работать со справочной и научной литературой, осуществлять поиск необходимой информации в Интернет;
- умение работать с техническими средствами;
- умение пользоваться соответствующими выполняемому проекту информационными технологиями;
- умение готовить материалы проекта для презентации: составлять и редактировать тексты, формировать презентацию проекта;
- умение работать в команде;

- умение публично представлять результаты собственной деятельности;
- коммуникабельность, инициативность, творческие способности.

### **Примерные вопросы для собеседования**

1. Моделирование биологических систем.
2. Автоматический компьютерный синтез речи по тексту. Методы синтеза речи.
3. Примеры систем обработки естественного языка.
4. Классификация инструментальных средств ЭС и организация знаний в ЭС.
5. Модели представления знаний: логическая, сетевая, фреймовая, продукционная.
6. Методы озвучивания речи.
7. История возникновения и современные направления исследований в области ИИ.
8. Классификация систем распознавания речи.
9. Этапы разработки экспертных систем.
10. Разработка прототипа экспертной системы.
11. Коллектив разработчиков интеллектуальных систем.
12. Распознавание символов. Шаблонные системы. Структурные системы. Признаковые системы.
13. Речевой вывод информации.
14. Машинный интеллект и робототехника.
15. Типы задач решаемые в ЭС.
16. Основные направления исследований в области искусственного интеллекта.
17. Предпосылки возникновения систем понимания естественного языка. Понимание в диалоге.
18. Распознавание рукописных текстов
19. Методы представления знаний: процедурные представления, логические представления, семантические сети, фреймы, системы продукций.
20. Интегрированные методы представления знаний.
21. Метазнания в системах ИИ.
22. Базы знаний.
23. Приобретение (извлечение) знаний.
24. Открытость знаний системы ИИ.
25. Машинное обучение: символьное обучение, генетические алгоритмы.

### **Примерные темы индивидуальных творческих проектов**

1. Проект «Электронный словарь»;
2. Проект «Экспертная система для диагностики здоровья»;
3. Проект «Лингвистическая экспертная система»;
4. Проект «Экспертная система для диагностики устройства»;
5. Проект «Обучающая экспертная система».

### **Контроль самостоятельной работы студентов**

Контроль самостоятельной работы проводится преподавателем в аудитории.  
Предусмотрены следующие виды контроля: собеседование.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

### **Список литературы для выполнения СРС**

#### **Основная литература:**

1. Пальмов С.В. Интеллектуальные системы и технологии Электронный ресурс: учебное пособие / С.В. Пальмов. - Самара: Поволжский государственный университет телекоммуникаций и информатики, 2017. - 195 с. - Книга находится в базовой версии ЭБС IPRbooks.

**Дополнительная литература:**

1. Смолин, Д.В. Введение в искусственный интеллект : конспект лекций / Д.В. Смолин. - М. : ФИЗМАТЛИТ, 2007. - 264 с.

2. Сотник С.Л. Проектирование систем искусственного интеллекта Электронный ресурс: учебное пособие / С.Л. Сотник. - Проектирование систем искусственного интеллекта, 2021-01-23. - Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. - 228 с. - Книга находится в базовой версии ЭБС IPRbooks.

**Методическая литература:**

1. Методические указания к практическим занятиям (электронный вариант)

2. Методические указания к самостоятельной работе (электронный вариант)

**Интернет-ресурсы:**

Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. – С-Пб.: Питер, 2000. [DJVU] (<http://www.twirpx.com/file/13533/>)

Мальковский М.Г. Конспект лекций по курсу «Искусственный интеллект» [HTML] (<http://al.cs.msu.su/classes/ai>).