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 студентов направления подготовки

05.03.03 Картография и геоинформатика
Геоинформатика

Ставрополь, 2026

СОДЕРЖАНИЕ

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 практических работ

№ Темы дисциплины	Наименование тем дисциплины, их краткое содержание	Объем часов	Из них практическая подготовка, часов
7 семестр			
1.	Анализ современного состояния области искусственного интеллекта.	**3**	**3**
2.	Знакомство с принципами функционирования искусственных нейронных сетей	**3**	**3**
3.	Принципы классификации образов с помощью искусственных нейронных сетей. Функции, массивы и объекты в Python	**6**	**6**
4.	Решение сложных задач классификации с помощью искусственных нейронных сетей. Распространение сигналов по нейронной сети.	**6**	**6**
5.	Решение сложных задач классификации с помощью искусственных нейронных сетей. Метод обратного распространения	**6**	**6**
6.	Обновление весовых коэффициентов. Набор рукописных цифр MNIST	**6**	**6**
7.	Тестирование нейронной сети для распознавания рукописных цифр MNIST	**6**	**6**
	Итого за 7 семестр	**36**	**36**
	Итого	**36**	**36**

СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ РАБОТ

Практическое занятие 1.
Анализ современных программных средств с применением ИИ

Цель занятия – сформировать представление об современных программных средств с применением ИИ.

Задачи занятия:

- изучить литературные источники и материалы из сети Интернет, связанные с нормативно-правовыми основами развития искусственного интеллекта в России и мире;
- проанализировать примеры применения технологий ИИ в образовательной сфере.

Задания для выполнения и методические рекомендации:

Задание 1.

Изучив литературные источники и материалы из сети Интернет, заполните таблицу «Нормативно-правовые основы развития искусственного интеллекта в России»:

№ п/п	Документ	Дата принятия	Ссылка на документ	Сроки реализации (если имеются)	Цели и задачи	Ответственные в реализации	Основные пункты
1.	«Национальная стратегия развития искусственного интеллекта на период до 2030 года»						
2.	…						

Задание 2.

Подготовьте сообщение (текст и презентацию) о любом из перечисленных в таблице Задания 1 документов. Будьте готовы рассказать его группе.

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.			
		1.2.			
		…			
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	Разработчик технологии	Время появления технологии	Описание возможностей технологии	Примеры использования технологии в профессиональной сфере
1.					
2.					

Задание 5.

Подготовьте сообщение (текст и презентацию) о любом из перечисленных в таблице Задания 4 технологий. Будьте готовы рассказать его группе.

Сделайте **вывод** о проделанной работе.

Рекомендуемая 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.

2. Сысоев, Д. В. Введение в теорию искусственного интеллекта : учебное пособие / Д. В. Сысоев, О. В. Курипта, Д. К. Проскурин. — Москва : Ай Пи Ар Медиа, 2021. — 170 с. — ISBN 978-5-4497-1092-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/108282.html

8.1.2. Перечень дополнительной литературы:

1. Аверченков, В. И. Система формирования знаний в среде Интернет : Монография / Аверченков В. И. - Брянск : Брянский государственный технический университет, 2012. - 181 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-89838-328-X

2. Павлов, С. И. Системы искусственного интеллекта : учебное пособие : [16+] / С. И. Павлов. – Томск : Томский государственный университет систем управления и радиоэлектроники, 2011. – Часть 1. – 175 с. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=208933

Интернет-ресурсы:

http://www.iprbookshop.ru - ЭБС IPRbooks
http://biblio-online.ru/ - ЭБС «Biblio-online.ru» издательства «Юрайт» ONLINE»
http://www.intuit.ru - Интернет-университет технологий

Практическое занятие 2
Знакомство с принципами функционирования искусственных нейронных сетей и с языком программирования Python

Цель: рассмотреть принципы функционирования искусственных нейронных сетей и познакомиться с языком программирования Python.

Задание:

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.

Теоретическое введение

1. Введение

2. Принципы функционирования искусственных нейронных сетей

1. Введение

Компьютеры в общем смысле – это калькуляторы, способные выполнять арифметические операции с огромной скоростью. Эта особенность компьютеров позволяет им отлично справляться с задачами, аналогичными тем, которые решаются с помощью калькуляторов: суммирование чисел с целью определения объемов продаж, применение процентных ставок для начисления налогов или построение графиков на основе существующих данных.

Даже просмотр телевизионных программ или прослушивание потоковой музыки через Интернет с помощью компьютера не требует чего-то большего, чем многократное выполнение простых арифметических операций. Реконструкция видеокадра, состоящего сплошь из единиц и нулей, которые поступают на ваш компьютер по сети, также осуществляется путем выполнения арифметических действий, лишь ненамного более сложных, чем суммирование чисел.

Сложение чисел с гигантской скоростью – тысячи или миллионы операций в секунду – это впечатляющий эффект, но его нельзя назвать проявлением искусственного интеллекта. Даже если человеку трудно складывать в уме большие числа, данный процесс вовсе не требует особого интеллекта. Для таких вычислений достаточно способности следовать элементарным инструкциям, и именно это происходит внутри любого компьютера.

Выводы

• У многих компьютерных систем имеются каналы ввода и вывода, между которыми над данными выполняются некоторые вычисления. В случае нейронных сетей имеет место быть другая структура работы.

• Если точные принципы функционирования какой-либо системы нам неизвестны, то мы пытаемся получить представление о том, как она работает, используя модель с регулируемыми параметрами. Если бы мы не знали, как преобразовать километры в мили, то могли бы использовать для этой цели линейную функцию в качестве модели с регулируемым наклоном.

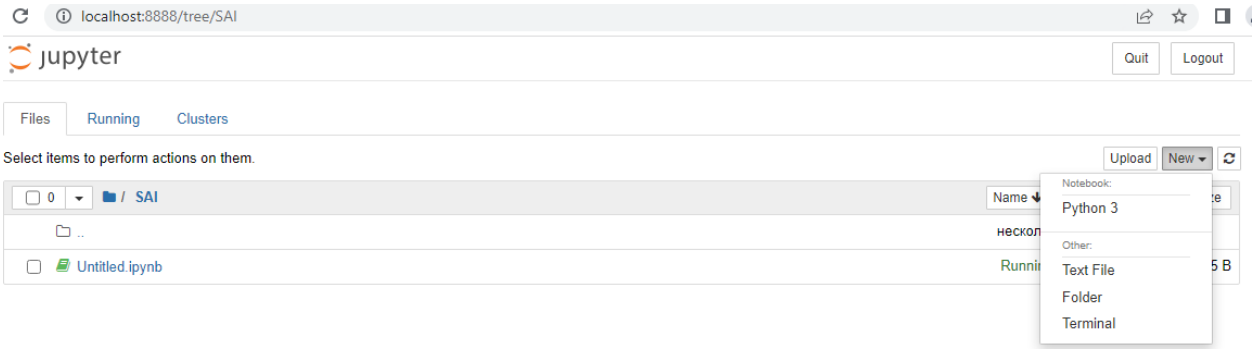
• Неплохим способом улучшения подобных моделей является настройка параметров на основании сравнения результатов модели с точными результатами в известных примерах.

Практические задания

1. Знакомство с Jupyter Notebook
2. Начало работы с Python
3. Многократное выполнение команд
4. Создание комментариев

1. Знакомство с Jupyter Notebook

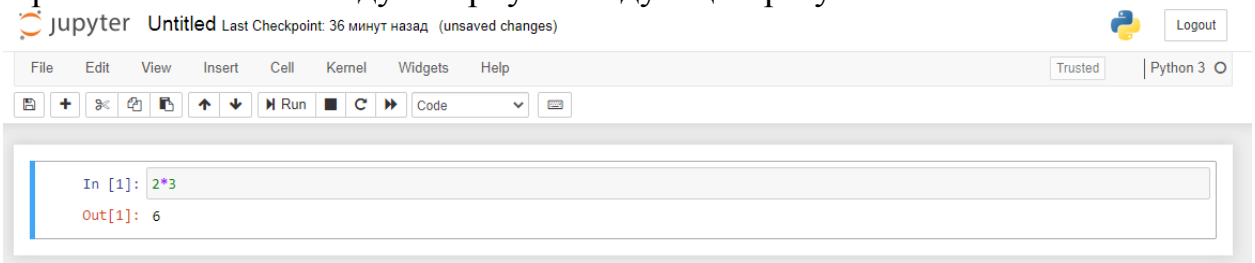
Запустив интерактивную оболочку Jupyter Notebook (Блокнот), щелкните на кнопке New у правого края окна, выберите в открывшемся меню пункт **Folder,** в названии папки укажите **свою фамилию.** Затем перейдите в созданную папку, щелкните на кнопке New у правого края окна и выберите **Python 3**, что приведет к открытию пустого **блокнота.**



Блокнот интерактивен, т.е. ожидает ввода команды, выводит ответ и вновь переходит в состояние ожидания следующей команды или вопроса.

При решении задач даже средней сложности имеет смысл разбивать их на части. Так легче структурировать логику задачи, а если что-то пойдет не так, особенно в большом проекте, вам будет проще найти ту его часть, в которой произошла ошибка. В терминологии IPython такие части называются **ячейками** (cells). В показанном выше блокноте ячейка пуста, и вы, наверное, заметили мерцающий курсор, который приглашает вас ввести в нее какую-нибудь команду.

Давайте прикажем компьютеру что-то сделать. Например, попросим его перемножить два числа, скажем, умножить 2 на 3. Введите в ячейку текст «2*3» (кавычки вводить не следует) и щелкните на кнопке run cell (выполнить ячейку), напоминающей кнопку воспроизведения в проигрывателе. Компьютер должен быстро выполнить команду и вернуть следующий результат.



Как видите, компьютер выдал правильный результат: «6». Только что мы предоставили компьютеру нашу первую инструкцию и успешно получили корректный ответ. Это была наша первая компьютерная программа!

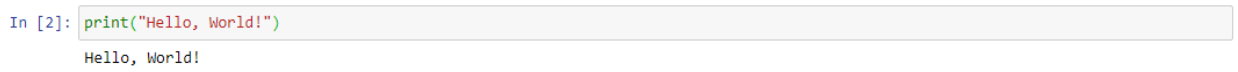
Не обращайте внимания на надписи «In [1]» и «Out[l]', которыми в IPython помечаются инструкция и ответ. Так оболочка напоминает о том, что именно вы просили сделать (input) и что вы получаете в ответ (output). Числа в скобках указывают на последовательность вопросов и ответов, что весьма удобно, когда вы то и дело вносите в код исправления и заново выполняете инструкции.

2. 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!»



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 удается 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.

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 [ ]:
```

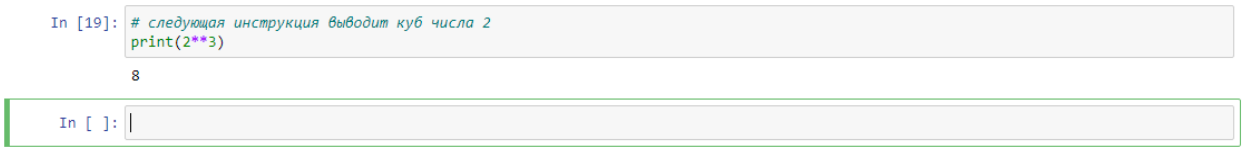
Точно так же можно увеличить число итераций до пятидесяти или тысячи с помощью выражений range (50) или range (1000). Попробуйте сделать это самостоятельно.

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.



Контрольные вопросы

1. В чем заключаются особенности вычислительных операций, выполняемых компьютером?
2. Кратко опишите принципы функционирования искусственных нейронных сетей.
3. С помощью какой команды можно вывести на печать Вашу фамилию на языке Python?
4. Как вывести список чисел от 0 до 20 на языке Python?
5. Как задаются комментарии на языке Python?

Практическое занятие 3

Принципы классификации образов с помощью искусственных нейронных сетей. Функции, массивы и объекты в Python.

Цель: изучить принципы классификации образов с помощью искусственных нейронных сетей и рассмотреть функции, массивы и объекты в Python.

Задание:

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:*

Отчѐт предоставляется в электронном виде в текстовом редакторе MSWord. В отчѐте 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 отчѐту, представленному студентом и с демонстрацией задания на компьютере.

Теоретическое введение

1. Задача классификации с помощью искусственных нейронных сетей
2. Тренировка простого классификатора

Задача классификации с помощью искусственных нейронных сетей

В прошлой лабораторной работе в теоретической части мы описали функционирование простой прогнозирующей машины. Она получает входные

данные и делает определенный прогноз относительно того, какими должны быть выходные данные. Мы улучшали прогнозы, регулируя внутренний параметр на основании величины ошибки, которую определяли, сравнивая прогноз с известным точным значением.

Далее на простом примере рассмотрим **задачу классификации** с помощью искусственных нейронных сетей.

Приведем диаграмму, представляющую результаты измерения размеров садовых жуков.

Выводы:

• Чтобы понять соотношение между выходной ошибкой линейного классификатора и параметром регулируемого наклона, достаточно владеть элементарной математикой. Зная это соотношение, можно определить величину изменения наклона, необходимую для устранения выходной ошибки.

• Недостаток прямолинейного подхода к регулировке параметров заключается в том, что модель обновляется до наилучшего соответствия только последнему тренировочному примеру, тогда как все предыдущие примеры не принимаются во внимание. Одним из неплохих способов устранения этого недостатка является уменьшение величины обновлений с помощью коэффициента скорости обучения, чтобы ни один отдельно взятый тренировочный пример не доминировал в процессе обучения.

• Тренировочные примеры, взятые из реальной практики, могут быть искажены шумом или содержать ошибки. Сглаживание обновлений способствует ограничению влияния подобных ложных примеров.

Практические задания

1. Функции
2. Массивы
3. Графическое представление массивов
4. Объекты

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 записыватьобразующий 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)
```

p ('второе число равно', y)a = (x + y) /2.0
r print('среднее значение равно', a)return a
i Обсудим, что делают эти команды. Первые две строки, начинающиеся
n
t
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 напомнят 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. Никакиечисла 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 длясоздания 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 установлениясоответствия 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:

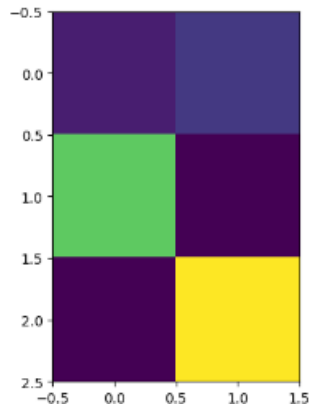
```
matplotlib.pyplot.imshow(a, interpolation="nearest")
```

```
[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 лаятьdef 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 снабжаетсяименем. 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 данномслучае 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!”

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()
```

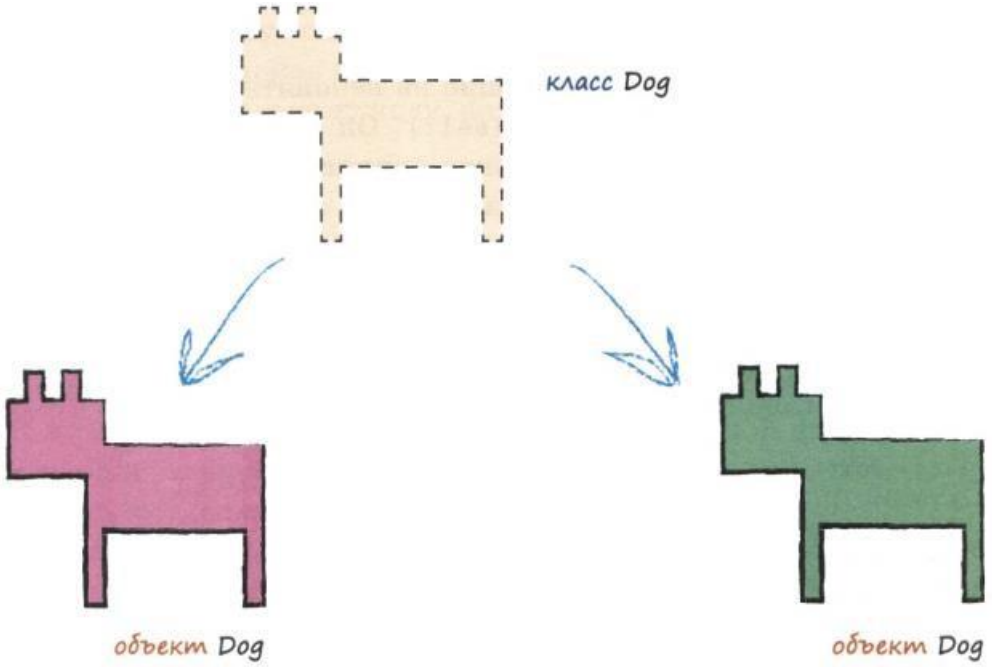
```
[16]: sizzles = Dog ()
      mutley = Dog()

      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.



А что нам дает создание объектов на основе класса? К чему все эти хлопоты? Не лучше ли было просто вывести фразу “Гав!” без какого-либо дополнительного кода?

Прежде всего, это имеет смысл делать тогда, когда возникает необходимость в создании множества однотипных объектов. Создавая эти объекты по единому образцу на основе класса, а не описывая полностью каждый из них по отдельности, вы экономите массу времени. Но реальное преимущество объектов заключается в том, что они обеспечивают компактное объединение данных и функциональности в одном месте. Централизованная организация фрагментов кода в виде объектов, которым они естественным образом принадлежат, значительно облегчает понимание структуры программы, особенно в случае сложных задач, что является большим плюсом для программистов. Собаки лают. Кнопки реагируют на

щелчки. Динамики издают звуки. Принтеры печатают или жалуются, если закончилась бумага. Во многих компьютерных системах кнопки, динамики и принтеры действительно являются объектами, функции которых вы активизируете.

Функции, принадлежащие объектам, называют методами. С включением функций в состав объектов вы уже сталкивались, когда мы добавляли функцию bark () в определение класса Dog, и созданные на основе этого класса объекты *sizzles* и *mutley* включали в себя данный метод. Вы видели, что оба они “лаяли” в рассмотренном примере.

Нейронные сети принимают некоторые входные данные (входной сигнал), выполняют некоторые вычисления и выдают выходные данные (выходной сигнал). Вы также знаете, что их можно тренировать. Вы уже видели, что эти действия – тренировка и выдача ответа – являются естественными функциями нейронной сети, т.е. их можно рассматривать в качестве функций объекта нейронной сети. С нейронными сетями естественным образом связаны относящиеся к ним внутренние данные – весовые коэффициенты связей между узлами. Вот почему мы будемсоздавать нашу нейронную сеть в виде объекта.

Чтобы вы получили более полное представление о возможностях объектов, давайте добавим в класс переменные, предназначенные для хранения специфических данных конкретных объектов, а также методы, позволяющие просматривать и изменять эти данные.

Рассмотрим ниже обновленное определение класса Dog. Оно включает ряд новых элементов, которые мы представим по отдельности.

```
# определение класса объектов Dogclass Dog:
# метод для инициализации объекта внутренними даннымиdef__init__(self,
petname, temp) :
self.name = petname; self.temperature = temp;
# получить состояниedef status (self) :
print("имя собаки: ", self.name) print("температура собаки: ",
self.temperature)pass
# задать температуру
def  setTemperature(self,  temp): self.temperature
= temp;pass
# собаки могут лаятьdef bark (self):
print("Гав!")pass
pass
```

Прежде всего, обратим внимание на то, что мы добавили в класс Dog три новые функции. У нас уже была функция bark(), а теперь мыдополнительно включили в класс функции *__init__(), status()* и

setTemperature(). Процедура добавления новых функций достаточно очевидна.
Рассмотрим переменные, указанные в круглых скобках после имен новых функций.,например, функция *setTemperature* фактически определена как *setTemperature (self, temp)*. Эти переменные, получения значений которых функции ожидают во время вызова, называются **параметрами.**
Вспомним функцию вычисления среднего *avg (x, y)*, с которой вы уже сталкивались. Определение функции *avg ()* явно указывало на то, что функция ожидает получения двух параметров. Следовательно, функция *__init___()* нуждается в параметрах *petname* и *temp*, а функция *setTemperature()* – только в параметре *temp*.
Заглянем внутрь этих функций. Начнем с функции с названием *__init_()*. Это специальное имя, и Python будет вызывать функцию *init ()* каждый раз при создании нового объекта данного класса. Это очень удобно для выполнения любой подготовительной работы до фактического использования объекта. Что же именно происходит в функции инициализации? Мы создаем две новые переменные: *self.name* и *self.temperature*. Вы можете узнать их значения из переменных *petname* и *temp*, передаваемых функции. Часть *self.* в именах означает, что эти переменные являются собственностью объекта, т.е. принадлежат данномуконкретному объекту и не зависят от других объектов Dog или общих переменных Python. Мы не хотим смешивать имя данной собаки с именем другой. Поначалу это может показаться слишком сложным, однако, все значительно упростится, когда мы приступим к рассмотрению конкретного примера.
Следующая – функция *status( )*, которая действительно проста. Она не принимает никаких параметров и просто выводит значения переменных *name*и *temperature* объекта Dog.
Наконец, функция *setTemperature ()* принимает параметр temp, значение которого при ее вызове присваивается внутренней переменной self.temperature. Это означает, что даже после того, как объект создан, вы можете в любой момент изменить его температуру, причем это можно сделать столько раз, сколько потребуется. Мы не будем тратить время на обсуждение того, почему все эти функции, включая *bark (),* принимают атрибут *self* в качестве первого параметра. Это особенность Python. По замыслу разработчиков это должно напоминать Python, что функция, которую вы собираетесь определить, принадлежит объекту, ссылкой на который служит *self* .
Чтобы прояснить все, о чем мы говорили, рассмотрим конкретный пример. В приведенном ниже коде вы видите обновленный класс Dog, определенный с новыми функциями, и новый объект *lassie*, создаваемый с параметрами, один из которых задает его имя как “Lassie”, а другой устанавливает его температуру равной 37. Как видите, вызов функции status()для объекта *lassie* класса *Dog* обеспечивает вывод его имени и текущего

значения температуры. С момента создания объекта эта температура не изменялась.

```
[19]: # определение класса объектов Dog
class Dog:
    # метод для инициализации объекта внутренними данными
    def __init__ (self, petname, temp) :
        self.name = petname;
        self.temperature = temp;
    # получить состояние
    def status (self) :
        print("имя собаки: ", self.name)
        print("температура собаки: ", self.temperature)
        pass
    # задать температуру
    def setTemperature(self, temp):
        self.temperature = temp;
        pass
    # собаки могут лаять
    def bark (self):
        print("Гав!")
        pass
    pass
```

```
[20]: # создать новый объект собаки на основе класса Dog
lassie=Dog("Lassie", 37)
```

```
[22]: lassie.status()
имя собаки:  Lassie
температура собаки:  37
```

Попытаемся изменить температуру объекта и проверим, действительноли она изменилась, введя следующий код.
lassie.setTemperature(40)
lassie.status()
Результат представлен ниже.

```
[22]: lassie.status()
имя собаки:  Lassie
температура собаки:  37

[24]: lassie.setTemperature(40)
lassie.status()
имя собаки:  Lassie
температура собаки:  40
```

Как видите, вызов функции setTemperature (40) действительно изменилвнутреннее значение температуры объекта.

Контрольные вопросы

1. В чем заключается задача классификации с помощью искусственных нейронных сетей?
2. Что представляют собой тренировочные данные?
3. Что понимают под сглаживанием при обучении искусственных нейронных сетей? С какой целью оно применяется?
4. Приведите этапы обучения сети в задаче классификации.
5. Что такое функция? Как задаются функции на языке Python?
6. Что включает и для чего используется пакет *numpy*?
7. Для каких задач используется пакет *matplotlib.pyplot*?
6. Как задать массив и вывести его графическое представление на языке Python?
7. Что такое класс, объект, метод?
8. Как задаются объекты на языке Python?

Практическое занятие 4

Решение сложных задач классификации с помощью искусственных нейронных сетей. Распространение сигналов понейронной сети

Цель: изучить возможности искусственных нейронных сетей для решения сложных задач классификации образов.

Задание:

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:*

Отчѐт предоставляется в электронном виде в текстовом редакторе MSWord. В отчѐте 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 отчѐту, представленному студентом и с демонстрацией задания на компьютере.

Теоретическое введение

1. Применение нескольких классификаторов
2. Нейроны – вычислительные машины, созданные природой
3. Распространение сигналов по нейронной сети

Практические задания

1. Создание класса нейронной сети
2. Инициализация сети
3. Весовые коэффициенты – основная часть сети
4. Улучшенный вариант инициализации весовых коэффициентов

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.

[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)
```

Данный код позволяет получить объект сети, но такой объект пока что не будет особенно 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, $\mathbf{W}_{\text{входной\_скрытый}}$, размерностью **hidden_nodes x input_nodes**;

• другую матрицу для связей между скрытым и выходным слоями, $\mathbf{W}_{\text{скрытый\_выходной}}$, 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 онлайновую документацию по функциям Python и находят полезные функции. При желании выполните такой поиск для функции numpy.random,rand (), еслихотите 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.

```
[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]])
```

Данный подход можно улучшить, поскольку весовые коэффициенты могут иметь 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 рассматривалиразличные 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 нулеи со стандартным отклонением, величина которого обратно пропорциональнакорню 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 параметрамиявляются 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

```
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))
```

отклонение 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.

Контрольные вопросы

1. Для каких задач применим простой линейный классификатор?
2. Каким образом произвести классификацию, если линейный классификатор не применим к задаче?
3. При каких значениях аргументов функция И принимает значение 1?
4. При каких значениях аргументов функция ИЛИ является истинной.

5. При каких значениях аргументов функция *исключающее ИЛИ* принимает значение 1?

6. Опишите строение нейрона.

7. Что представляет собой функция активации?

8. Какая функция чаще всего используется в качестве функции активации в искусственных нейронных сетях? По каким причинам?

9. Что такое весовые коэффициенты сети?

10. Каким образом происходит распространение сигнала по искусственной нейронной сети?

11. Какие функции может содержать класс нейронной сети, созданный на языке Python?

Практическое занятие 5
Решение сложных задач классификации с помощью искусственных нейронных сетей. Метод обратного распространения

Цель: рассмотреть метод обратного распространения ошибки приразработке проекта нейронной сети.

Задание:

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:*

Отчѐт предоставляется в электронном виде в текстовом редакторе MSWord. В отчѐте 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 отчѐту, представленному студентом и с демонстрацией задания на компьютере.

Теоретическое введение

1. Умножение матриц
2. Пример использования матричного умножения в сети с тремя слоями
3. Корректировка весовых коэффициентов в процессе обучения нейронной сети
4. Обратное распространение ошибок при большом количестве слоев
5. Описание обратного распространения ошибок с помощью матричной алгебры

Резюме

- Нейронные сети обучаются посредством уточнения весовых коэффициентов своих связей. Этот процесс управляется ошибкой – разностью между правильным ответом, предоставляемым тренировочными данными, и фактическим выходным значением.
- Ошибка на выходных узлах определяется простой разностью между желаемым и фактическим выходными значениями.
- В то же время величина ошибки, связанной с внутренними узлами, не столь очевидна. Одним из способов решения этой проблемы является распределение ошибок выходного слоя между соответствующими связями пропорционально весу каждой связи с последующим объединением

соответствующих разрозненных частей ошибки на каждом внутреннем узле.

Практические задания

1. Опрос сети
2. Текущее состояние кода
3. Тренировка сети

Проект нейронной сети на Python. Часть 2.

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 междусоответствующими узлами, а так□же 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 сигмоиды, 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:

$$\mathbf{X}_{\text{скрытый}} = \mathbf{W}_{\text{входной\_скрытый}} * \mathbf{I}$$

Компьютерные языки, в том числе Python, распознают матрицы и эффективно выполняют все расчеты, поскольку им известно об однотипностивсех стоящих за этим вычислений.
Код на языке Python для этой операции достаточно простой. Нижепредставлена 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 при□меняем к каждому из них сигмоиду:

$$\mathbf{O}_{\text{скрытый}} = \text{сигмоида}\,(\mathbf{X}_{\text{скрытый}})$$

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 томчисле 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 водном 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 естьне что иное, как сигмоида. Функции, создаваемые с помощью лямбда- выражений, являются безымянными или, как предпочитают говорить опытные программисты, анонимными, но данной функции мы присвоилиимя *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, вы□деленные 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:
    # wll w21
    # wl2 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 сигмоиды 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 двухмерный 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 IPython следует поместить код, импортирующий модули *numpy* и *scipy.special*.

Функции *query ()* в качестве входных данных потребуются только входные

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

сигналы *input_list*. Ни в каких других входных данных она не нуждается.

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([[0.638757861],
            [0.656137261],
            [0.436785441]])
```

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__()*, 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 двухмерный 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:

$$\text{ошибки}_{\text{скрытый}} = \text{веса}^{T}_{\text{скрытый\_выходной}} * \text{ошибки}_{\text{выходной}}$$

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 ивыходным слоями мы используем переменную *output_errors*.

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

Ранее нами было получено выражение для обновления веса связи между узлом **j** и узлом **k** следующего слоя в матричной форме.

$$\Delta W_{jk} = \alpha * E_k * \text{сигмоида}(O_k) * (1 - \text{сигмоида}(O_k)) \cdot O_j^T$$

Величина ***a*** – это коэффициент обучения, а сигмоида – это функция активации, с которой вы уже знакомы. Вспомните, что символ «*» означает обычное поэлементное умножение, а символ «·» – скалярное произведение матриц. Последний член выражения – это транспонированная (^{T}) матрица исходящих сигналов предыдущего слоя. В данном случае *транспонирование* означает преобразование столбца выходных сигналов в строку.

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))
```

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 поможетвам 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))
```

self.who += self.lr * numpy .dot ((output_errors * final_outputs * (1.0 - final_outputs)), numpy.transpose (hidden_outputs))

self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)), numpy.transpose(inputs))

Вот и все!

Вся та работа, для выполнения которой нам потребовалось множество вычислений, и все те усилия, которые мы вложили в разработку матричного подхода и способа минимизации ошибок сети методом градиентного спуска, свелись к паре строк кода! Отчасти мы обязаны этим языку Python, но фактически это закономерный результат нашего упорного труда, вложенногов упрощение того, что легко могло стать сложным и приобрести устрашающий вид.

Приведем полный код функции train. **Его необходимо добавить в программу.**

```
# тренировка нейронной сети
def train (self, inputs_list, targets_list) :
    # преобразование списка входных значений
    # в двухмерный 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)

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

    # 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))

    pass
```

Полный код нейронной сети может быть использован с целью создания, тренировки и опроса трехслойной нейронной сети практически для любой задачи.

Далее мы займемся решением одной конкретной задачи: обучение нейронной сети распознаванию рукописных цифр.

Контрольные вопросы

1. Что такое матрица?
2. Как получить скалярное произведение матриц? Приведите пример.
3. Что означает транспонирование матриц?
4. Что представляют собой входной, выходной и скрытый слои нейронной сети?
5. Опишите кратко процесс корректировки весовых коэффициентов в процессе обучения нейронной сети.
6. Что такое метод обратного распространения ошибки?

Практическое занятие 6
Обновление весовых коэффициентов. Набор рукописных цифр MNIST.

Цель: рассмотреть процесс обновления весовых коэффициентов; подготовить тренировочные данные MNIST для проекта нейронной сети по распознаванию рукописных цифр.

Задание:

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:*

Отчѐт предоставляется в электронном виде в текстовом редакторе MSWord. В отчѐте 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 отчѐту, представленному студентом и с демонстрацией задания на компьютере.

Теоретическое введение

Рассмотрение процесса обновления весовых коэффициентов

Практические задания

1. Набор рукописных цифр MNIST
2. Подготовка тренировочных данных MNIST

1. Набор рукописных цифр MNIST

Распознавание текста, написанного от руки, – это настоящий вызов для испытания возможностей искусственного интеллекта, поскольку эта проблема действительно трудна и размыта. Она не столь ясная и четко определенная, как перемножение одного множества чисел на другое.

Корректная классификация содержимого изображений с помощью компьютера, которую часто называют **распознаванием образов**, десятилетиями выдерживала атаки, направленные на ее разрешение. В последнее время в этой области наблюдается значительный прогресс, и решающую роль в наметившемся прорыве сыграли технологии нейронных сетей.

О трудности проблемы можно судить хотя бы по тому, что даже мы, люди, иногда не можем договориться между собой о том, какое именноизображение мы видим. В частности, предметом спора может послужить и написанная неразборчивым почерком буква.

Существует коллекция изображений рукописных цифр, используемых исследователями искусственного интеллекта в качестве популярного набора для тестирования идей и алгоритмов.

Этим тестовым набором является база данных рукописных цифр под названием «MNIST», предоставляемая авторитетным исследователем нейронных сетей Яном Лекуном для бесплатного всеобщего доступа по адресу http://yann.lecun.com/exdb/mnist/. Там же вы найдете сведения относительно успешности прежних и нынешних попыток корректного распознавания этих рукописных символов.

Формат базы данных MNIST не относится к числу тех, с которыми легко работать, но, к счастью, другие специалисты создали соответствующие файлы в более простом формате (см., например, http://pjreddie.com/projects/mnist-in-csv/). Это так называемые CSVфайлы, вкоторых отдельные значения представляют собой обычный текст и разделены запятыми. Их содержимое можно легко просматривать в любом текстовом редакторе, и большинство электронных таблиц или программ, предназначенных для анализа данных, могут работать с CSV-файлами. Это довольно универсальный формат. На указанном сайте предоставлены следующие два файла:

•тренировочный набор (http://www.pjreddie.com/media/files/mnist_train.csv);

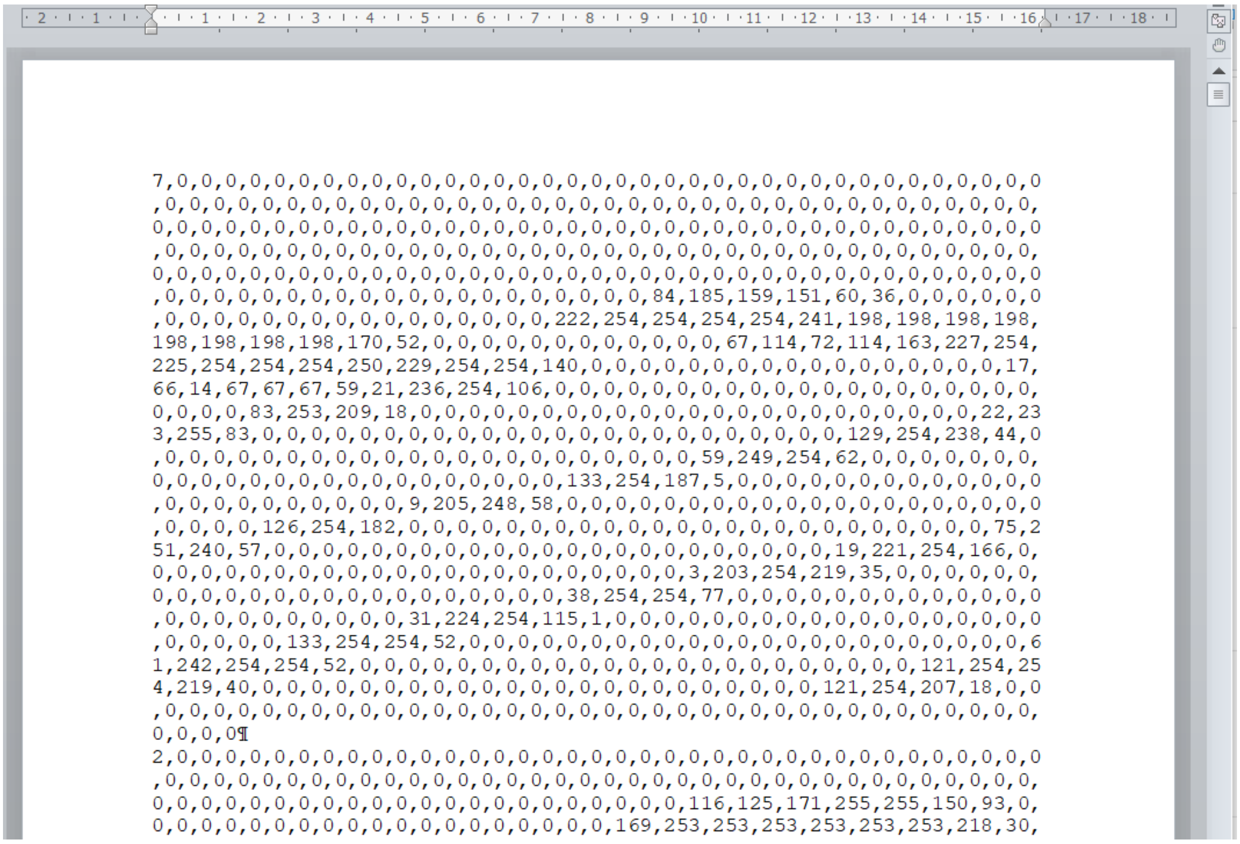
• тестовый набор (http://www.pjreddie.com/media/files/mnist_test.csv).

Тренировочный набор содержит **60 000** промаркированных экземпляров, используемых для тренировки нейронной сети. Слово «промаркированные» означает, что для каждого экземпляра указан соответствующий правильный ответ.

Меньший **тестовый набор**, включающий **10 000** экземпляров, используется для проверки правильности работы идей или алгоритмов. Он также содержит корректные маркеры, позволяющие увидеть, способна ли наша нейронная сеть дать правильный ответ.

Использование независимых друг от друга наборов тренировочных и тестовых данных гарантирует, что с тестовыми данными нейронная сеть ранее не сталкивалась. В противном случае можно было бы схитрить и просто запомнить тренировочные данные, чтобы получить наибольшие баллы. Идея разделения тренировочных и тестовых данных распространена среди специалистов по машинному обучению.

Присмотримся к этим файлам. Ниже показана часть тестового набора MNIST, загруженного в текстовый редактор.



```
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,84,185,159,151,60,36,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,222,254,254,254,254,241,198,198,198,198,
198,198,198,198,170,52,0,0,0,0,0,0,0,0,0,0,0,0,67,114,72,114,163,227,254,
225,254,254,254,250,229,254,254,140,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,17,
66,14,67,67,67,59,21,236,254,106,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,83,253,209,18,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,22,23
3,255,83,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,129,254,238,44,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,59,249,254,62,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,133,254,187,5,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,9,205,248,58,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,126,254,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,75,2
51,240,57,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,19,221,254,166,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,203,254,219,35,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,38,254,254,77,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,31,224,254,115,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,133,254,254,52,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6
1,242,254,254,52,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,254,25
4,219,40,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,254,207,18,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0¶
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,116,125,171,255,255,150,93,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,169,253,253,253,253,253,253,218,30,
```

В окне текстового редактора отображаются длинные строки текста (в конце каждой строки отображается знак абзаца), которые содержат числа,

разделенные запятыми. Это легко можно увидеть. Текстовые строки такие длинные, потому что каждая из них занимает несколько строк на экране.
Содержимое этих записей, т.е. строк текста, легко понять.

• Первое значение – это **маркер**, т.е. фактическая цифра, например «7» или «9» , которую должен представлять данный рукописный экземпляр. Это ответ, правильному получению которого должна обучиться нейронная сеть.

• Последующие значения, разделенные запятыми, – это значения пикселей рукописной цифры. Пиксельный массив имеет размерность 28x28, поэтому за каждым маркером следуют 784 пикселя.

Таким образом, первая запись представляет цифру «7» , о чем говорит первое значение, тогда как остальная часть текста в этой строке – это пиксельные значения написанной кем-то от руки цифры «7» . Вторая строка представляет рукописную цифру «2» , третья – цифру «4» , четвертая –цифру « 1» и пятая – цифру «9» . Можете выбрать любую строку из файлов данных MNIST, и ее первое число укажет вам маркер для последующихданных изображения.
Однако увидеть, каким образом длинный список из 784 значений формирует изображение рукописной цифры «7» , не так-то просто. Мы должны вывести в графическом виде эти цифры и убедиться в том, что они действительно представляют цвета пикселей рукописной цифры.
Прежде чем углубиться в рассмотрение деталей и приступить к написанию кода, загрузим небольшое подмножество набора данных, содержащихся в базе MNIST. Файлы данных MNIST имеют очень большой размер, тогда как работать с небольшими наборами значительно удобнее, поскольку это позволит нам

экспериментировать, разрабатывать и испытывать свой код, что было бы затруднительно в случае длительных расчетов из-за большого объема обрабатываемых данных. Когда мы отработаем алгоритм, можно будет вернуться к полному набору данных.
Прежде чем с этими данными можно будет что-либо сделать, например, построить график или обучить с их помощью нейронную сеть, необходимо обеспечить доступ к ним из кода на языке Python.
Открытие файлов и получение их содержимого в Python не составляет большого труда. Лучше всего показать, как это делается, на конкретном примере. Взгляните на следующий код.

```
data_file = open("mnist_train.csv", 'r')
data_list = data_file.readlines()
data_file.close()
```

Здесь всего три строки кода. Обсудим каждую из них по отдельности. Первая строка открывает файл с помощью функции open (). Вы видите,
что в качестве первого из параметров ей передается имя файла. На самом деле это не просто имя файла, а путь доступа к нему. Второй параметр необязательный и сообщает Python, как мы хотим работать с файлом. Буква «r» означает, что мы хотим открыть файл только для чтения, а не для записи. Тем самым мы предотвращаем любое непреднамеренное изменение или удаление файла. Если мы попытаемся осуществить запись в этот файл или изменить его, Python не позволит этого сделать и выдаст ошибку. А что этоза переменная **data_file**? Функция open () создает дескриптор (указатель), играющий роль ссылки на открываемый файл, и мы присваиваем его переменной data_file. Когда файл открыт, любые последующие действия с ним, например, чтение, осуществляются через этот дескриптор.
Следующая строка проще. Мы используем функцию readlines (), ассоциированную с дескриптором файла data_file, для чтения всех строк содержимого файла в переменную data_list. Эта переменная содержит список, каждый элемент которого является строкой, представляющей строку файла. Это очень удобно, поскольку мы можем переходить к любой строке файла так же, как к конкретным записям в списке. Таким образом, data_list[0]
– это первая запись, a data_list [9] – десятая и т.п.
Гораздо эффективнее работать поочередно с каждой строкой, а не считывать в память весь файл целиком. Однако наши файлы невелики, и использование функции readlines () значительно упрощает код, а для наспростота и ясность на данном этапе очень важны.
Последняя строка закрывает файл. Считается хорошей практикой закрывать файлы и другие ресурсы после их использования. Если же оставлять их открытыми, то это может приводить к возникновению проблем. Некоторые программы могут отказываться осуществлять запись в файл, открытый в другом месте, если это приводит к несогласованности. В противном случае это напоминало бы ситуацию, когда два человека пытаются одновременно записывать разные тексты на одном и том же листе бумаги. Иногда компьютер

может заблокировать файл во избежание такого рода конфликтов. Если не оставить за собой порядок после того, как необходимость в использовании файла уже отпала, у вас может накопиться много заблокированных файлов. По крайней мере, закрыв файл, вы предоставите компьютеру возможность освободить память, занимаемую уже ненужными данными.
Создайте новый пустой блокнот, выполните представленный ниже код и посмотрите, что произойдет, если попытаться вывести элементы списка.

```
In [11]:  data_file = open("mnist_train.csv", 'r')
          data_list = data_file.readlines()
          data_file.close()

In [12]:  len(data_list)

Out[12]:  60000

In [13]:  data_list[0]

Out[13]:  '5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,166,255,247,127,
0,0,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0,0,0,0,0,0,0,0,0,0,0,49,238,253,253,253,253,253,253,253,253,251,93,82,82,56,39,0,
0,0,0,0,0,0,0,0,0,0,0,18,219,253,253,253,253,253,198,182,247,241,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,11,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,14,1,154,253,90,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,81,240,253,253,119,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,25
3,253,150,27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,253,253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,
201,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,253,253,195,80,9,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,55,172,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,253,253,253,212,135,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\n'
```

Вы видите, что длина списка равна 6000. Функция Python 1en() сообщает о размере списка. Вы также можете видеть содержимое первой

записи, data_list[0]. Первое число, 5, – это маркер, тогда как остальные 784 числа – это цветовые коды пикселей, из которых состоит изображение. Если вы внимательно присмотритесь, то заметите, что их значения не выходят за пределы диапазона 0 до 255. Вы можете взглянуть на другие записи, чтобы проверить, выполняется ли это условие и для них. Вы убедитесь в том, чтотак оно и есть: значения кодов всех цветов попадают в диапазон чисел от 0до 255.
Ранее приводился пример графического отображения прямоугольного массива чисел с использованием функции imshow (). То же самое мы хотим сделать и в данном случае, но для этого нам нужно преобразовать список чисел, разделенных запятыми, в подходящий массив. Это можно сделать в соответствии со следующей процедурой:

- разбить длинную текстовую строку значений, разделенных запятыми, на отдельные значения, используя символ запятой в качестве разделителя;
- проигнорировать первое значение, являющееся маркером, извлечь оставшиеся 28*28=784 значения и преобразовать их в массив, состоящий из 28 строк и 28 столбцов;
- отобразить массив.

И вновь, проще всего показать соответствующий простой код на Pythonи уже после этого подробно рассмотреть, что в нем происходит.
Прежде всего, мы не должны забывать о необходимости импортировать библиотеки расширений Python для работы с массивами и графикой.
Посмотрите на следующие три строки кода. Переменные окрашеныразличными

```
import numpy
import matplotlib.pyplot
%matplotlib inline
```

цветами таким образом, чтобы было понятно, где и какие данные используются.

```
all_values = data_list[0].split(',')
image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

В первой строке длинная первая запись data_list[0], которую мы только что выводили, разбивается на отдельные значения с использованием запятой в качестве разделителя. Это делается с помощью функции split (), параметр которой определяет символ-разделитель. Результат помещается в переменную all_values. Можете вывести эти значения на экран и убедиться в том, что эта переменная действительно содержит нужные значения в виде длинного списка Python.

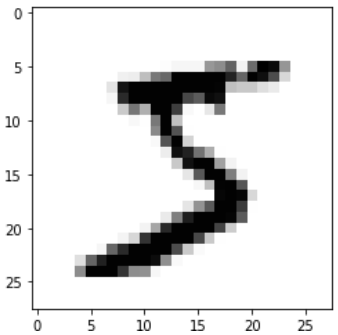
Следующая строка кода выглядит чуть более сложной, поскольку в ней происходит сразу несколько вещей. Начнем с середины. Запись списка в виде all_values [1:] указывает на то, что берутся все элементы списка за исключением первого. Тем самым мы игнорируем первое значение, играющее роль маркера, и берем лишь остальные 784 элемента,

numpy.asfarray() – это функция библиотеки numpy, преобразующая текстовые строки в реальные числа и создающая массив этих чисел. Но почему текстовые строки преобразуются в числа? Если файл был прочитан как текстовый, то каждая строка или запись все еще остается текстом. Извлечение из них отдельных элементов, разделенных запятыми, также дает текстовые элементы. Например, этим текстом могли бы быть слова « apple» ,
«orange123» или «567» . Текстовая строка «567» – это не то же самое, что число 567. Именно поэтому мы должны преобразовывать текстовые строки в числа, даже если эти строки выглядят как числа. Последний фрагмент инструкции – .reshape ((28,28)) – гарантирует, что список будет сформирован в виде квадратной матрицы размером 28x28. Результирующий массив такой же размерности получает название image_array.

Третья строка просто выводит на экран массив image_array с помощью функции imshow (), аналогично тому, с чем вы уже сталкивались. На этот раз мы выбрали цветовую палитру оттенков серого с помощью параметра cmap =' Greys' для лучшей различимости рукописных символов.

Результат работы кода представлен ниже.

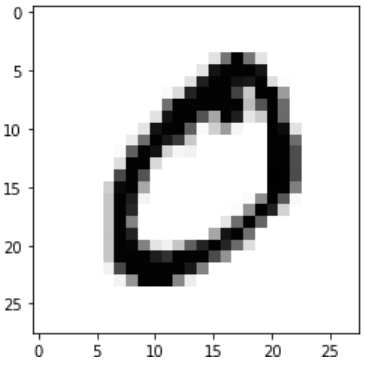
```
In [18]: all_values = data_list[0].split(',')
         image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
         matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
Out[18]: <matplotlib.image.AxesImage at 0x2aa4a9c1340>
```



Вы видите графическое изображение цифры «5» , на что и указывал маркер. Если мы выберем следующую запись, data_list [1] с маркером 0, то получим показанное ниже изображение.

```
In [19]: all_values = data_list[1].split(',')
         image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
         matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

```
Out[19]: <matplotlib.image.AxesImage at 0x2aa4b12c130>
```



Глядя на это изображение, вы с уверенностью можете сказать, что этой рукописной цифре действительно соответствует цифра 0.

2. Подготовка тренировочных данных MNIST

Когда мы уже знаем, как получить данные из файлов MNIST и извлечь из них нужные записи, мы можем воспользоваться этим для визуализации данных. Мы хотим использовать эти данные для обучения нашей нейронной сети, но сначала мы должны продумать, как их следует подготовить, прежде чем предоставлять сети.

Вы уже видели, что нейронные сети работают лучше, если как входные, так и выходные данные конфигурируются таким образом, чтобыони оставались в диапазоне значений, оптимальном для функций активации узлов нейронной сети.

Первое, что мы должны сделать, – это перевести значения цветовых кодов из большего диапазона значений 0-255 в намного меньший, охватывающий значения от 0,01 до 1,0. Мы намеренно выбрали значение0,01 в качестве нижней границы диапазона, чтобы избежать упомянутых ранее проблем с нулевыми входными значениями, поскольку они могут искусственно блокировать обновление весов. Нам необязательно выбирать значение 0,99 в качестве верхней границы допустимого диапазона, поскольку нет нужды избегать значений 1,0 для входных сигналов. Лишь выходные сигналы не могут превышать значение 1,0.

Деление исходных входных значений, изменяющихся в диапазоне 0- 255, на 255 приведет их к диапазону 0-1,0. Последующее умножение этих значений на коэффициент 0,99 приведет их к диапазону 0,0-0,99. Далее мы инкрементируем их на 0,01, чтобы вместить их в желаемый диапазон 0,01- 1,0. Все эти действия реализует следующий код на языке Python.

```
In [24]: # привести входные значения к диапазону 0,01 - 1,00
scaled_input = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
print(scaled_input)
```

```
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.208    0.62729412 0.99223529 0.62729412 0.20411765
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.19635294 0.934
0.98835294 0.98835294 0.98835294 0.93011765 0.01     0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.21964706 0.89129412 0.99223529 0.98835294 0.93788235
0.91458824 0.98835294 0.23129412 0.03329412 0.01     0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.01    0.01    0.01
0.01    0.01    0.01    0.04882353 0.24294118 0.87964706
0.98835294 0.99223529 0.98835294 0.79423529 0.33611765 0.98835294
0.99223529 0.48364706 0.01     0.01     0.01     0.01
```

Результирующий вывод данных подтверждает, что они действительно принадлежат к диапазону значений от 0,01 до 1,00.

Итак, мы осуществили подготовку данных MNIST путем их масштабирования и сдвига и теперь можем подавать их на вход нашей нейронной сети как с целью ее тренировки, так и с целью опроса.

На этом этапе нам также нужно продумать, что делать с выходными значениями нейронной сети. Ранее вы видели, что выходные значения должны укладываться в диапазон значений, обеспечиваемый функцией активации. Используемая нами логистическая функция не может выдавать такие значения, как -2,0 или 255. Ее значения охватывают диапазон чисел от 0,0 до 1,0, а фактически вы никогда не получите значений 0,0 или 1,0, поскольку логистическая функция не может их достигать и лишь асимптотически приближается к ним. Таким образом, по-видимому, нам придется масштабировать выходные значения в процессе тренировки сети.

Но что именно мы должны получить на выходе нейронной сети? Должно ли это быть изображение ответа? В таком случае нам нужно иметь 28x28=784 выходных узла.

Если мы вернемся на шаг назад и задумаемся над тем, чего именно мы хотим от нейронной сети, то поймем, что мы просим ее классифицировать изображение и присвоить ему корректный маркер. Таким маркером может быть одно из десяти чисел в диапазоне от 0 до 9. Это означает, что выходной слой сети должен иметь 10 узлов, по одному на каждый возможный ответ, или маркер. Если ответом является «0» , то активизироваться должен первый узел, тогда как остальные узлы должны оставаться пассивными. Если ответом является «9» , то активизироваться должен последний узел выходного слоя при пассивных остальных узлах. Следующая иллюстрация поясняет эту схему на нескольких примерах выходных значений.

выходной слой	маркер	пример "5"	пример "0"	пример "9"
0	0	0.00	0.95	0.02
1	1	0.00	0.00	0.00
2	2	0.01	0.01	0.01
3	3	0.00	0.01	0.01
4	4	0.01	0.02	0.40
5	5	0.99	0.00	0.01
6	6	0.00	0.00	0.01
7	7	0.00	0.00	0.00
8	8	0.02	0.00	0.01
9	9	0.01	0.02	0.86

Первый пример соответствует случаю, когда сеть распознала входные данные как цифру «5» . Вы видите, что наибольший из исходящих сигналов выходного слоя принадлежит узлу с меткой «5» . Не забывайте о том, что это шестой по счету узел, потому что нумерация узлов начинается с нуля. Все

довольно просто. Остальные узлы производят сигналы небольшой величины, близкие к нулю. Вывод нулей мог оказаться следствием ошибок округления, но в действительности, как вы помните, функция активации никогда не допустит фактическое нулевое значение.
Следующий пример соответствует рукописному «0» . Наибольшую величину здесь имеет сигнал первого выходного узла, ассоциируемый с меткой «0» .
Последний пример более интересен. Здесь самый большой сигнал генерирует последний узел, соответствующий метке «9» . Однако и узел с меткой «4» дает сигнал средней величины. Обычно нейронная сеть должна принимать решение, основываясь на наибольшем сигнале, но, как видите, в данном случае она отчасти считает, что правильным ответом могло бы бытьи «4» . Возможно, рукописное начертание символа затруднило его надежное распознавание? Такого рода неопределенности встречаются в нейронных сетях, и вместо того, чтобы считать их неудачей, мы должны рассматривать их как полезную подсказку о существовании другого возможного ответа.
Отлично! Теперь нам нужно превратить эти идеи в целевые массивы для тренировки нейронной сети. Как вы могли убедиться, если тренировочный пример помечен маркером «5» , то для выходного узла следует создать такой целевой массив, в котором малы все элементы, кроме одного, соответствующего маркеру «5» . В данном случае этот массив мог бывыглядеть примерно так: [0,0,0,0,0,1,0,0,0,0].
В действительности эти числа нуждаются в дополнительном масштабировании, поскольку мы уже видели, что попытки создания на выходе нейронной сети значений 0 и 1, недостижимых в силу использования функции активации, приводят к большим весам и насыщению сети. Следовательно, вместо этого мы будем использовать значения 0,01 и 0,99, и потому целевым массивом для маркера «5» должен быть массив [0.01, 0.01, 0.01, 0.01, 0.01, 0.99, 0.01, 0.01, 0.01, 0.01].
А вот как выглядит код на языке Python, создающий целевую матрицу.

```
# количество выходных узлов - 10 (пример)
onodes = 10
targets = numpy.zeros(onodes) + 0.01
targets[int(all_values[0])] =0.99
```

Первая строка после комментария просто устанавливает количествовыходных узлов равным 10, что соответствует нашему примеру с десятью маркерами.
Во второй строке с помощью удобной функции numpy.zeros () создается массив, заполненный нулями. Желаемые размер и конфигурация массива задаются параметром при вызове функции. В данном случае создается одномерный массив, размер *onodes* которого равен количеству узлов в конечном выходном слое. Проблема нулей, которую мы только что обсуждали, устраняется путем добавления 0,01 к каждому элементу массива.
Следующая строка выбирает первый элемент записи из набора данных MNIST, являющийся целевым маркером тренировочного набора, и

преобразует его в целое число. Вспомните о том, что запись читается из исходного файла в виде текстовой строки, а не числа. Как только преобразование выполнено, полученный целевой маркер используется для того, чтобы установить значение соответствующего элемента массива равным 0,99. Здесь все будет нормально работать, поскольку маркер «0»будет преобразован в целое число 0, являющееся корректным индексом данного маркера в массиве targets [ ]. Точно так же маркер «9» будет преобразован в целое число 9, и элемент targets [9] действительно является последним элементом этого массива. Вот пример работы этого кода.

```
In [23]: # количество выходных узлов - 10 (пример)
         onodes = 10
         targets = numpy.zeros(onodes) + 0.01
         targets[int(all_values[0])] =0.99

In [25]: print(targets)

         [0.99 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
```

Теперь мы знаем, как подготовить входные значения для тренировки иопроса нейронной сети, а выходные значения – для тренировки.

Обновим наш код с учетом проделанной работы. Ниже представленосостояние кода на данном этапе, включая последнее обновление.

```
In [1]: import numpy
        # библиотека scipy.special с сигмоидой expit ()
        import scipy.special
        # гарантировать размещение графики в данном блокноте,
        # а не в отдельном окне
        %matplotlib inline
```

In [2]:

```
# 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:
        # wil w21
        # wl2 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 сигмоиды в качестве функции активации
        self.activation_function = lambda x: scipy.special.expit(x)

        pass


    # тренировка нейронной сети
    def train (self, inputs_list, targets_list) :
        # преобразование списка входных значений
        # в двухмерный 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)
    # ошибки выходного слоя =
    # (целевое значение - фактическое значение)
    output_errors = targets - final_outputs
    # 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)

    # 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))

    pass


# опрос нейронной сети
def query(self, inputs_list):
    # преобразовать список входных значений
    # в двухмерный 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
```

In [3]:
```
input_nodes = 784
hidden_nodes = 100
output_nodes = 10

# коэффициент обучения равен 0,3
learning_rate = 0.3

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

In [5]:
```
# загрузить в список тестовый набор данных CSV-файла набора MNIST
training_data_file = open("mnist_train.csv", 'r')
training_data_list = training_data_file.readlines ()
training_data_file.close()

# тренировка нейронной сети

# перебрать все записи в тренировочном наборе данных
for record in training_data_list:
  # получить список значений, используя символы запятой (' , ')
  # в качестве разделителей
  all_values = record.split(',')
  # масштабировать и сместить входные значения
  inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
  # создать целевые выходные значения (все равны 0,01, за исключением
  # желаемого маркерного значения, равного 0,99)
  targets = numpy.zeros(output_nodes) + 0.01

  # all_values[0] - целевое маркерное значение для данной записи
  targets[int(all_values[0])] =0.99
  n.train(inputs, targets)
  pass
```

В самом начале этого кода мы импортируем графическую библиотеку, добавляем код для задания размеров входного, скрытого и выходного слоев, считываем малый тренировочный набор данных MNIST, а затем тренируем нейронную сеть с использованием этих записей.

Почему мы выбрали 784 входных узла? Вспомните о том, что это числоравно произведению 28x28, представляющему количество пикселей, из которых состоит изображение рукописной цифры.

Выбор ста скрытых узлов не имеет столь же строгого научного обоснования. Мы не выбрали это число большим, чем 784, из тех соображений, что нейронная сеть должна находить во входных значениях такие особенности или шаблоны, которые можно выразить в более короткой форме, чем сами эти значения. Поэтому, выбирая количество узлов меньшим, чем количество входных значений, мы заставляем сеть пытаться находить ключевые особенности путем обобщения информации. В то же время, если выбрать количество скрытых узлов слишком малым, будут ограничены возможности сети в отношении определения достаточного количества отличительных признаков или шаблонов в изображении. Тем самым мы лишили бы сеть возможности выносить собственные суждения относительно данных MNIST. С учетом того, что выходной слой должен обеспечиватьвывод 10 маркеров, а значит, должен иметь десять узлов, выбор промежуточного значения 100 для количества узлов скрытого слоя представляется вполне разумным.

В связи с этим следует сделать одно важное замечание: идеального общего метода для выбора количества скрытых узлов не существует. В действительности не существует и идеального метода выбора количества скрытых слоев. В настоящее время наилучшим подходом является

проведение экспериментов до тех пор, пока не будет получена конфигурациясети, оптимальная для задачи, которую вы пытаетесь решить.

Контрольные вопросы

1. В чем заключается метод градиентного спуска?
2. Назовите преимущества метода градиентного спуска?
3. Что означает функция ошибки (ошибка)? Как подсчитать ошибку?
4. Что представляют собой коэффициентом обучения нейронной сети?
5. Каким образом происходит обновление весовых коэффициентов в процессе обучения нейронной сети.
6. Что представляют собой набор рукописных цифр MNIST?
7. Что такое распознавание образов?
8. Что такое тренировочный набор?
9. Что такое тестовый набор?

Практическое занятие 7
Тестирование нейронной сети для распознавания рукописных цифр MNIST.

Цель: провести тестирование разработанной нейронной сети для распознавания рукописных цифр MNIST.

Задание:

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:*

Отчѐт предоставляется в электронном виде в текстовом редакторе MSWord. В отчѐте 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 отчѐту, представленному студентом и с демонстрацией задания на компьютере.

Тестирование нейронной сети

В данной работе проверим, как работает, и сделаем нейросеть на тестовом наборе данных.

Прежде всего, необходимо получить тестовые записи. Соответствующий код очень похож на тот, который мы использовали для получения тренировочных данных.

```
# загрузить в список тестовый набор данных CSV-файла набора MNIST
test_data_file = open("mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
```

Мы распакуем эти данные точно так же, как и предыдущие, поскольку они имеют аналогичную структуру.

Прежде чем создавать цикл для перебора всех тестовых записей, посмотрим, что произойдет, если мы вручную выполним одиночный тест. Ниже представлены результаты опроса уже обученной нейронной сети, выполненного с использованием первой записи тестового набора данных.

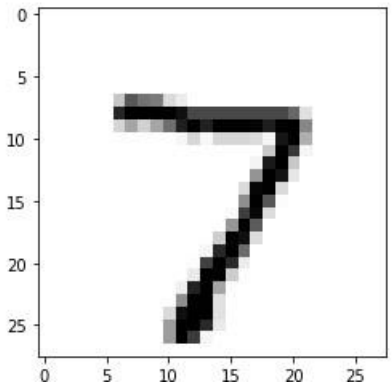
```
# загрузить в список тестовый набор данных CSV-файла набора MNIST
test_data_file = open("mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
```

```
# получить первую тестовую запись
all_values = test_data_list[0].split(',')
#  вывести маркер
print(all_values[0])
```

```
7
```

```
image_array = numpy.asfarray(all_values[1:]).reshape((28,28))
matplotlib.pyplot.imshow(image_array, cmap='Greys', interpolation='None')
```

```
29]: <matplotlib.image.AxesImage at 0x251210f4b20>
```



Как видите, в качестве маркера первой записи тестового набора сеть определила символ “ 7” .

Не забудьте подключить библиотеку *matplotlib.pyplot*.

```
import numpy
# библиотека scipy.special с сигмоидой expit ()
import scipy.special
# гарантировать размещение графики в данном блокноте,
# а не в отдельном окне
import matplotlib.pyplot
%matplotlib inline
```

В результате опроса обученной сети мы получаем список чисел, являющихся выходными значениями каждого из выходных узлов. Сразу же бросается в глаза, что одно из выходных значений намного превышает остальные, и этому значению соответствует маркер “7” . Это восьмой элемент списка, поскольку первому элементу соответствует маркер “0” .

У нас все сработало! Мы обучили нашу нейронную сеть и добились того, что она смогла определить цифру, предоставленную ей в виде изображения. Вспомните, что до этого сеть не сталкивалась с данным изображением, поскольку оно не входило в тренировочный набор данных. Следовательно, нейронная сеть оказалась в состоянии корректно классифицировать незнакомый ей цифровой символ.

Далее напишем код, позволяющий проверить, насколько хорошо нейронная сеть справляется с остальной частью набора данных, и провести подсчет правильных результатов, чтобы впоследствии мы могли оценивать плодотворность своих будущих идей по совершенствованию способности

сети обучаться, а также сравнивать наши результаты с результатами, полученными другими людьми.
Ознакомьтесь с приведенным ниже кодом.

```
# тестирование нейронной сети
# журнал оценок работы сети, первоначально пустой
scorecard = []
# перебрать все записи в тестовом наборе данных
for record in test_data_list:
  # получить список значений из записи, используя символы
  # запятой (',') в качестве разделителей
  all_values = record.split(',')
  # правильный ответ - первое значение
  correct_label = int(all_values[0])
  print(correct_label, "истинный маркер")
  # масштабировать и сместить входные значения
  inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
  # опрос сети
  outputs = n.query(inputs)
  # индекс наибольшего значения является маркерным значением
  label = numpy.argmax(outputs)
  print(label, "ответ сети")
  # присоединить оценку ответа сети к концу списка
  if (label == correct_label) :
    # в случае правильного ответа сети присоединить
    # к списку значение 1
    scorecard.append(1)
  else:
    # в случае неправильного ответа сети присоединить
    # к списку значение 0
    scorecard.append(0)
    pass
  pass
```

Разберем данный код. Прежде чем войти в цикл, обрабатывающий все записи тестового набора данных, мы создаем пустой список *scorecard*, который будет служить нам журналом оценок работы сети, обновляемым после обработки каждой записи.
В цикле мы делаем то, что уже делали раньше: извлекаем значения из текстовой записи, в которой они разделены запятыми. Первое значение, указывающее правильный ответ, сохраняется в отдельной переменной. Остальные значения масштабируются, чтобы их можно было использовать в качестве входных данных для передачи запроса нейронной сети. Ответ нейронной сети сохраняется в переменной *outputs*.
Далее следует довольно интересная часть кода. Мы знаем, чтонаибольшее из значений выходных узлов рассматривается сетью в качестве правильного ответа. Индекс этого узла, т.е. его позиция, соответствует маркеру. Эта фраза просто означает, что первый элемент соответствует маркеру “0” , пятый – маркеру “4” и т.д. К счастью, существует удобная функция библиотеки numpy, которая находит среди элементов массива максимальное значение и сообщает его индекс. Это функция *numpy.argmax ()*. Для получения более подробных сведений о ней можете посетить веб-страницу по следующему адресу:
*https: / /docs.scipy.org/doc/numpy-1.10.1/reference/*generated/numpy.argmax.html

Возврат этой функцией значения 0 означает, что правильным ответом сеть считает “0” и т.д.
В последнем фрагменте кода полученный маркер сравнивается с известным корректным маркером. Если оба маркера одинаковы, в журнал записывается “1” , в противном случае – “0” .

```
7 истинный маркер
7 ответ сети
2 истинный маркер
2 ответ сети
1 истинный маркер
1 ответ сети
0 истинный маркер
0 ответ сети
4 истинный маркер
4 ответ сети
1 истинный маркер
1 ответ сети
4 истинный маркер
4 ответ сети
9 истинный маркер
9 ответ сети
5 истинный маркер
5 ответ сети
```

Кроме того, в некоторых местах кода можно включить полезную команду *print ()*, чтобы мы сами могли отслеживать правильные и предсказываемые значения. Ниже представлены результаты выполнения этого кода вместе с выведенными записями нашего рабочего журнала.

```
print (scorecard)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

Дополним код фрагментом, который будет выводить относительную долю правильных ответов в виде дроби.

```
# рассчитать показатель эффективности в виде
# доли правильных ответов
scorecard_array = numpy.asarray(scorecard)
print ("эффективность = ", scorecard_array.sum() / scorecard_array.size)
```

```
эффективность =  0.9466
```

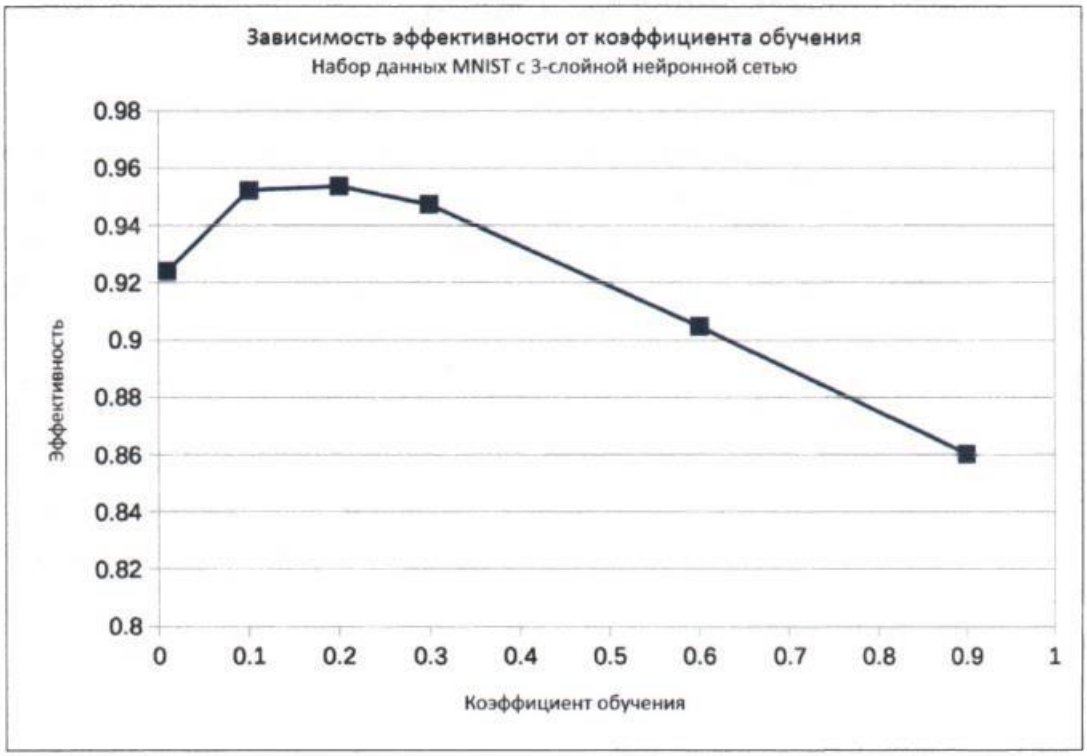
Эта доля рассчитывается как количество всех записей в журнале, содержащих “1” , деленное на общее количество записей (размер журнала). Вот каким получается результат.
В соответствии с результатами обучения нашей простой трехслойной нейронной сети с использованием полного набора, включающего 60 тысяч

примеров, и последующего тестирования на 10 тысячах записей показатель общей эффективности сети составляет 0,9466.
Этот показатель, равный почти 95%, можно сравнить с аналогичными результатами эталонных тестов, которые можно найти по адресу http://yann.lecun.com/exdb/mnist/. Вы увидите, что в некоторых случаях наши результаты даже лучше эталонных и почти сравнимы с приведенными на указанном сайте результатами для простейшей нейронной сети.
Это вовсе не так плохо. Мы должны быть довольны тем, что наша первая попытка продемонстрировала эффективность на уровне нейронной сети профессиональных исследователей.
Кстати, вас не должно удивлять, что даже в случае современных быстродействующих домашних компьютеров обработка всех 60 тысяч тренировочных примеров, для каждого из которых необходимо вычислить распространение сигналов от 784 входных узлов через сто скрытых узлов в прямом направлении, а также обратное распространение ошибок и обновление весов, занимает ощутимое время.

Улучшение результатов: настройка коэффициента обучения

Попытаемся улучшить разработанный к этому моменту код, внеся в него некоторые усовершенствования.
Прежде всего, мы можем попытаться настроить коэффициент обучения. Перед этим мы задали его равным 0,3, даже не тестируя другие значения.
Давайте удвоим это значение до величины 0,6 и посмотрим, как это скажется на способности нейронной сети обучаться. Выполнение кода с таким значением коэффициента обучения дает показатель эффективности 0,9047. Этот результат хуже прежнего. По-видимому, увеличение коэффициента обучения нарушает монотонность процесса минимизации ошибок методом градиентного спуска и сопровождается перескоками через минимум.
Повторим вычисления, установив коэффициент обучения равным 0,1. На этот раз показатель эффективности улучшился до 0,9523, что почти совпадает с результатом эталонного теста, который проводился с использованием тысячи скрытых узлов.
Что произойдет, если мы пойдем еще дальше и уменьшим коэффициент обучения до 0,01? Оказывается, это также приводит к уменьшению показателя эффективности сети до 0,9241. По-видимому, слишком малые значения коэффициента обучения снижают эффективность. Это представляется логичным, поскольку малые шаги уменьшают скорость градиентного спуска.
Описанные результаты представлены ниже в виде графика. Дляполучения более точных научных результатов нам следовало бы провести такие эксперименты множество раз, чтобы исключить фактор случайности и влияние неудачного выбора маршрутов градиентного спуска, но в целом он

позволяет проиллюстрировать общую идею о существовании некоего оптимального значения коэффициента обучения.



Вид графика подсказывает нам, что оптимальным может быть значение в интервале от 0,1 до 0,3, поэтому испытаем значение 0,2. Показатель эффективности получится равным 0,9537. Мы видим, что этот результат немного лучше тех, которые мы имели при значениях коэффициента обучения, равных 0,1 или 0,3.

Итак, мы остановимся на значении коэффициента обучения 0,2, которое проявило себя как оптимальное для набора данных MNIST и нашей нейронной сети.

При выполнении этого кода ваши оценки будут немного отличаться от приведенных, поскольку процесс в целом содержит элементы случайности. Ваш случайный выбор начальных значений весовых коэффициентов не будет совпадать с приведенным, а потому маршрут градиентного спуска для вашего кода будет другим.

Улучшение результатов: многократное повторение тренировочных циклов

Следующим усовершенствованием будет многократное повторение циклов тренировки с одним и тем же набором данных. В отношении одного тренировочного цикла иногда используют термин эпоха. Поэтому сеанс тренировки из десяти эпох означает десятикратный прогон всего тренировочного набора данных. А зачем нам это делать?

Особенно если для этого компьютеру потребуется 10, 20 или даже 30 минут? Причина заключается в том, что тем самым мы обеспечиваем большее число маршрутов градиентного спуска, оптимизирующих весовые коэффициенты.

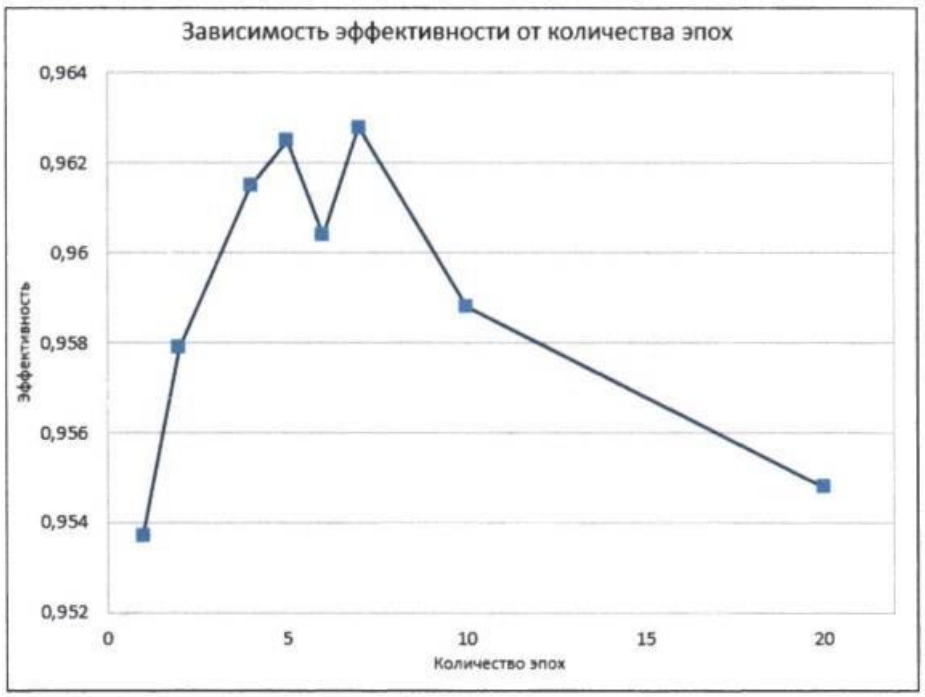
Посмотрим, что нам дадут две тренировочные эпохи. Для этого мы должны немного изменить код, предусмотрев в нем дополнительный цикл выполнения кода тренировки.

```
# тренировка нейронной сети
# переменная epochs указывает, сколько раз тренировочный
# набор данных используется для тренировки сети
epochs = 2
for e in range(epochs):
  # перебрать все записи в тренировочном наборе данных
  for record in training_data_list:
    # получить список значений из записи, используя символы
    # запятой (',') в качестве разделителей
    all_values = record.split(',')
    # масштабировать и сместить входные значения
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
    # создать целевые выходные значения (все равны 0,01, за
    # исключением желаемого маркерного значения, равного 0,99)
    targets = numpy.zeros(output_nodes) + 0.01
    # all_values[0] - целевое маркерное значение для
    # данной записи
    targets[int(all_values[0])] =0.99
    n.train(inputs, targets)
    pass
  pass
```

Результирующий показатель эффективности для двух эпох составляет 0,9579, что несколько лучше показателя для одной эпохи.

Подобно тому, как мы настраивали коэффициент обучения, проведем эксперимент с использованием различного количества эпох и построим график зависимости показателя эффективности от этого фактора. Интуиция подсказывает нам, что чем больше тренировок, тем выше эффективность. Но можно предположить, что слишком большое количество тренировок чревато ухудшением эффективности из-за так называемого **переобучения** сети на тренировочных данных, снижающего эффективность при работе с незнакомыми данными. Фактора переобучения следует опасаться в любых видах машинного обучения, а не только в нейронных сетях.

В данном случае мы имеем следующие результаты.

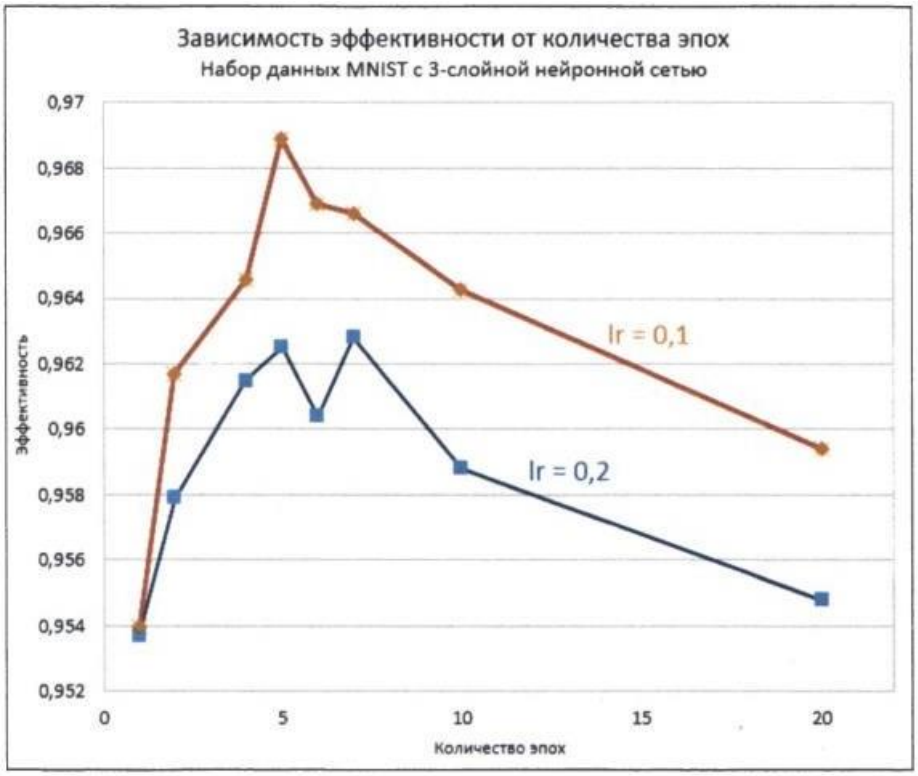


Теперь пиковое значение эффективности составляет 0,9628, или 96,28%, при семи эпохах.

Как видите, результаты оказались не столь предсказуемыми, как ожидалось. Оптимальное количество эпох – 5 или 7. При больших значениях эффективность падает, что может быть следствием переобучения. Провалпри 6 эпохах, по всей вероятности, обусловлен неудачными параметрами цикла, которые привели градиентный спуск к ложному минимуму. На самом деле можно было ожидать большей вариации результатов, поскольку мы не проводили множества экспериментов для каждой точки данных, чтобы уменьшить вариацию, вызванную случайными факторами. Именно поэтому мы оставили эту странную точку, соответствующую шести эпохам, чтобы напомнить вам, что по самой своей сути обучение нейронной сети – это случайный процесс, который иногда может работать не очень хорошо, а иногда и очень плохо.

Другое возможное объяснение – это то, что коэффициент обученияоказался слишком большим для большего количества эпох. Повторим эксперимент, уменьшив коэффициент обучения с 0,2 до 0,1, и посмотрим, чтопроизойдет.

На следующем графике новые значения эффективности при коэффициенте обучения 0,1 представлены вместе с предыдущими результатами, чтобы их было легче сравнить между собой.



Как нетрудно заметить, уменьшение коэффициента обучения действительно привело к улучшению эффективности при большомколичестве эпох. Пиковому значению 0,9689 соответствует вероятность ошибок, равная 3% , что сравнимо с эталонными результатами, указанными на сайте Яна Лекуна по адресу http://yann .lecun.com/exdb/mnist/.

Интуиция подсказывает нам, что если мы планируем использовать метод градиентного спуска при значительно большем количестве эпох, то уменьшение коэффициента обучения (более короткие шаги) в целом приведет к выбору лучших маршрутов минимизации ошибок. Вероятно, 5 эпох – это оптимальное количество для тестирования нашей нейронной сетис набором данных MNIST.

Изменение конфигурации сети

Один из факторов, влияние которых мы еще не исследовали, – конфигурация нейронной сети. Необходимо попробовать изменить количество узлов скрытого промежуточного слоя. Мы установили его равным 100.

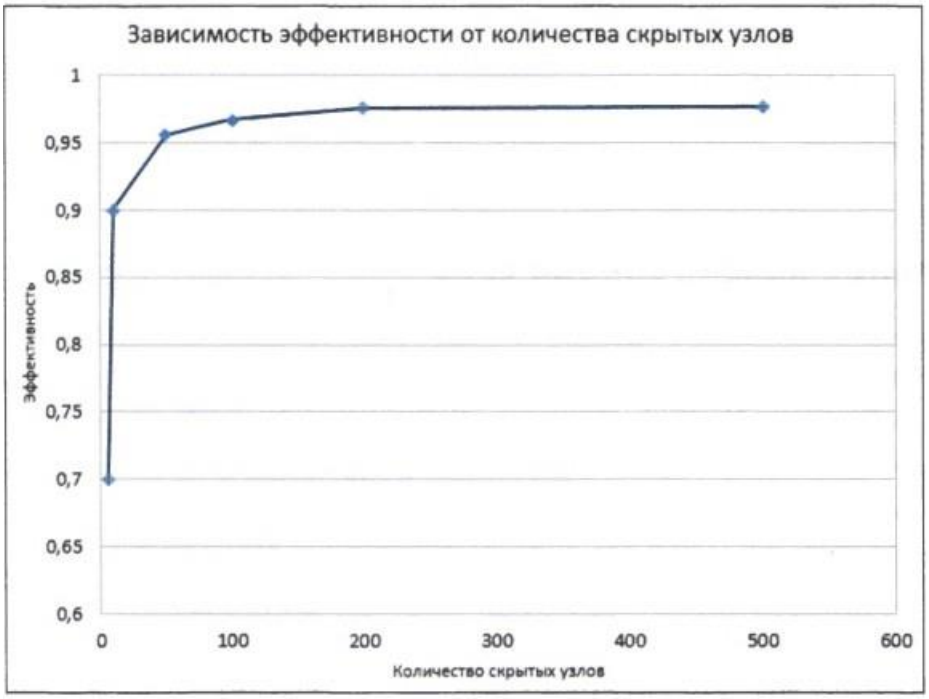
Прежде чем приступить к экспериментам с различными количествами скрытых узлов, давайте подумаем, что при этом может случиться. Скрытый слой как раз и является тем слоем, в котором происходит обучение. Вспомните, что узлы входного слоя принимают входные сигналы, а узлы выходного выдают ответ сети. Собственно, скрытый слой (или слои) должен научить сеть превращать входные сигналы в ответ. Именно в нем происходитобучение.

Если бы у нас было слишком мало скрытых узлов, скажем, три, то мы не имели бы никаких шансов обучить сеть чему-либо, научить ее

преобразовывать все входные сигналы в корректные выходные сигналы. Ограничения такого рода называют **способностью к обучению**. Невозможно обучить большему, чем позволяет способность к обучению, но можно увеличить способность к обучению, изменив конфигурацию сети.

Что, если бы у нас было 10 тысяч скрытых узлов? Ну да, в таком случае способности к обучению вполне хватило бы, но мы могли бы столкнуться с трудностями обучения сети, поскольку число маршрутов обучения было бы непомерно большим. Не исключено, что для тренировки такой сети понадобилось бы 10 тысяч эпох.

Выполним эксперименты и посмотрим, что получится.



Как видите, при небольшом количестве узлов результаты хуже, чем при больших количествах. Это совпадает с нашими ожиданиями. Однако даже для всего лишь пяти скрытых узлов показатель эффективности составил0,7001. Это удивительно, поскольку при столь малом количестве узлов, в которых происходит собственно обучение сети, она все равно дает 70% правильных ответов. Вспомните, что до этого мы выполняли тесты, используя сто скрытых узлов. Но даже десять скрытых узлов обеспечивают точность 0,8998, или 90%, что тоже впечатляет.

Нейронная сеть способна давать хорошие результаты даже при небольшом количестве скрытых узлов, в которых и происходит обучение, что свидетельствует о ее мощных возможностях.

По мере дальнейшего увеличения количества скрытых узлов результаты продолжают улучшаться, однако уже не так резко. При этом значительно растет время, затрачиваемое на тренировку сети, поскольку добавление каждого дополнительного скрытого узла приводит к увеличению количества связей узлов скрытого слоя с узлами предыдущего и следующего слоев, а вместе с этим и объема вычислений. Поэтому необходимо достигать

компромисса между количеством скрытых узлов и допустимыми затратами времени.

Мы также установили новый рекорд точности: 0,9751 при 200 скрытых узлах. Длительный расчет с 500 узлами обеспечил точность 0,9762. Это действительно неплохо по сравнению с результатами эталонных тестов, опубликованными на сайте Лекуна.

Возвращаясь к графикам, можно заметить, что неподдающийся ранее предел точности 97% был преодолен за счет изменения конфигурации сети.

Итоги

Давая общую оценку проделанной нами работы, необходимоподчеркнуть, что при создании нейронной сети с помощью Python мы использовали лишь самые простые концепции.

И тем не менее с помощью нейронной сети нам, даже без использования каких-либо дополнительных математических ухищрений, удалось получить вполне достойные результаты.

Далее можно продолжить эксперименты с уже созданной нами нейронной сетью – **используйте другое количество скрытых узлов, задавайте другие коэффициенты масштабирования или даже задействуйте другую функцию активации и анализируйте, что при этом происходит.**

Рекомендуемая литература:

1. Интеллектуальные информационные системы и технологии / Ю.Ю. Громов. - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2013. - 244 с. - ISBN 978-5-8265-1178-7

2. Матвеев, М. Г. Модели и методы искусственного интеллекта. Применение в экономике / М.Г. Матвеев ; А.С. Свиридов ; Н.А. Алейникова. - Москва : Финансы и статистика, 2011. - 448 с. - ISBN 978-5-279-03279-2

3. 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. Аверченков, В. И. Система формирования знаний в среде Интернет : Монография / Аверченков В. И. - Брянск : Брянский государственный технический университет, 2012. - 181 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-89838-328-X

2. Богомолова, М.А.
    Экспертные системы (техника и технология проектирования) Электронный ресурс : учебно-методическое пособие / М.А. Богомолова. - Самара : Поволжский государственный университет телекоммуникаций и информатики, 2015. - 47 с. - Книга находится в базовой версии ЭБС IPRbooks.

3. 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.

4. Ясницкий, Л. Н. Введение в искусственный интеллект : учебное пособие для студентов вузов, обучающихся по мат. направлениям и специальностям / Л.Н. Ясницкий. - 3-е изд., стер. - М. : Академия, 2010. - 176 с. : ил. - (Высшее профессиональное образование. Информатика и вычислительная техника). - Библиогр.: с. 170-173. - ISBN 978-5-7695-7042-1

Интернет-ресурсы:

1. Воройский Ф. С. Информатика. Энциклопедический словарь-справочник: введение в современные информационные и телекоммуникационные технологии в терминах и фактах. - М.: ФИЗМАТЛИТ, 2006. - 768 с. – Доступно: http://physics-for-students.ru/bookpc/informatika/slovar.zip

2. Иванов В. Основы искусственного интеллекта – https://libtime.ru/expertsystems/osnovy-iskusstvennogo-intellekta.html

3. Романов П.С. Основы искусственного интеллекта; Учебно-метод. пособие. – http://www.studfiles.ru/preview/2264160/

4. Сайт Основы ИИ – https://sites.google.com/site/osnovyiskusstvennogointellekta/

5. Соболь Б.В. Информатика: учебник/ Б.В. Соболь [и др.] – Изд. 3-е, дополн. и перераб. – Ростов н/Д: Феникс, 2007. – 446 с. – Доступно: http://physics-for-students.ru/bookpc/informatika/Sobol.rar

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»

05.03.03 Картография и геоинформатика
направленность (профиль):
Геоинформатика

Ставрополь, 2025

СОДЕРЖАНИЕ

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 стремясь 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 согласоваться 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 Практическое занятие, 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 согласоваться 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.

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком профессиональном уровне. Представленный материал в основном фактически верен, допускаются негрубые фактические неточности. Студент свободно отвечает на вопросы, связанные с проектом.	Технические средства и информационные технологии освоены и использованы для реализации проекта полностью	Студент проявил инициативу, творческий подход, способность к выполнению сложных заданий, навыки работы в коллективе, организационные способности.	Проект представлен полностью и в срок.
«Хорошо»	Работа выполнена на достаточно высоком профессиональном уровне. Допущено до 4–5 фактических ошибок. Студент отвечает на вопросы, связанные с проектом, но недостаточно полно.	Обнаруживаются некоторые ошибки в использовании соответствующих технических средств и информационных технологий	Студент достаточно полно, но без инициативы и творческих находок выполнил возложенные на него задачи.	Проект представлен достаточно полно и в срок, но с некоторыми недоработками.
«Удовлетворительно»	Уровень недостаточно высок. Допущено до 8 фактических ошибок. Студент может ответить лишь на некоторые из заданных вопросов,	Обнаруживает недостаточное владение навыками работы с техническими средствами и соответствующим и	Студент выполнил большую часть возложенной на него работы.	Проект сдан со значительным опозданием (более недели) и не полностью

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
	связанных с проектом.	информационным и технологиями		
«Неудовлетворительно»	Работа не выполнена или выполнена на низком уровне. Допущено более 8 фактических ошибок. Ответы на связанные с проектом вопросы обнаруживают непонимание предмета и отсутствие ориентации в материале проекта.	Навыков работы с техническими средствами нет, информационные технологии не освоены	Студент практически не работал, не выполнил свои задачи или выполнил лишь отдельные не существенные поручения в групповом проекте.	Проект не сдан.

Студенты должны: защитить проект в режиме презентации, предъявить файлы выполненного проекта, уметь рассказать о технологиях, использованных ими при выполнении проекта, дать оценку работы каждого члена группы (*если проект групповой*).

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».

3.6. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий, особенно по математике - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неутомительные занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Правила подготовки к зачетам и экзаменам:

• Лучше сразу сориентироваться во всем материале и обязательно расположить весь материал согласно экзаменационным вопросам (или вопросам, обсуждаемым на семинарах), эта работа может занять много времени, но все остальное – это уже технические детали (главное – это ориентировка в материале!).

• Сама подготовка связана не только с «запоминанием». Подготовка также предполагает и переосмысление материала, и даже рассмотрение альтернативных идей.

• Готовить «шпаргалки» полезно, но пользоваться ими рискованно. Главный смысл подготовки «шпаргалок» – это систематизация и оптимизация знаний по данному предмету, что само по себе прекрасно – это очень сложная и важная для студента работа, более сложная и важная, чем простое поглощение массы учебной информации. Если студент самостоятельно подготовил такие «шпаргалки», то, скорее всего, он и экзамены сдавать будет более уверенно, так как у него уже сформирована общая ориентировка в сложном материале.

• Как это ни парадоксально, но использование «шпаргалок» часто позволяет отвечающему студенту лучше демонстрировать свои познания (точнее – ориентировку в знаниях, что намного важнее знания «запомненного» и «тут же забытого» после сдачи экзамена).

• Сначала студент должен продемонстрировать, что он «усвоил» все, что требуется по программе обучения (или по программе данного преподавателя), и лишь после этого он вправе высказать иные, желательно аргументированные точки зрения.

4. Контроль самостоятельной работы магистрантов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка реферата, оценка презентации, оценка участия в круглом столе, оценка выполнения проекта.

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

5. Список литературы для выполнения СРС

Рекомендуемая 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 c. - Книга находится в базовой версии ЭБС IPRbooks.

2. Сысоев, Д. В. Введение в теорию искусственного интеллекта : учебное пособие / Д. В. Сысоев, О. В. Курипта, Д. К. Проскурин. — Москва : Ай Пи Ар Медиа, 2021. — 170 c. — ISBN 978-5-4497-1092-5. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/108282.html

8.1.2. Перечень дополнительной литературы:

1. Аверченков, В. И. Система формирования знаний в среде Интернет : Монография / Аверченков В. И. - Брянск : Брянский государственный технический университет, 2012. - 181 c. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 5-89838-328-X

2. Павлов, С. И. Системы искусственного интеллекта : учебное пособие : [16+] / С. И. Павлов. – Томск : Томский государственный университет систем управления и радиоэлектроники, 2011. – Часть 1. – 175 с. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=208933

3. Павлов, С. И. Системы искусственного интеллекта : учебное пособие / С. И. Павлов. – Томск : Томский государственный университет систем управления и радиоэлектроники, 2011. – Часть 2. – 194 с. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=208939

4. Сергеев, Н. Е. Системы искусственного интеллекта : учебное пособие : [16+] / Н. Е. Сергеев. – Таганрог : Южный федеральный университет, 2016. – Часть 1. – 123 с. : схем., ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=493307

5. Сотник, С. Л. Проектирование систем искусственного интеллекта : учебное пособие / С. Л. Сотник. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2021. — 228 c. — ISBN 978-5-4497-0868-7. — Текст : электронный // Цифровой образовательный ресурс IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/102054.html.

Интернет-ресурсы:

1. Воройский Ф. С. Информатика. Энциклопедический словарь-справочник: введение в современные информационные и телекоммуникационные технологии в терминах и фактах. - М.: ФИЗМАТЛИТ, 2006. - 768 с. – Доступно: http://physics-for-students.ru/bookpc/informatika/slovar.zip

2. Иванов В. Основы искусственного интеллекта – https://libtime.ru/expertsystems/osnovy-iskusstvennogo-intellekta.html

3. Романов П.С. Основы искусственного интеллекта; Учебно-метод. пособие. – http://www.studfiles.ru/preview/2264160/

4. Сайт Основы ИИ – https://sites.google.com/site/osnovyiskusstvennogointellekta/

5. Соболь Б.В. Информатика: учебник/ Б.В. Соболь [и др.] – Изд. 3-е, дополн. и перераб. – Ростов н/Д: Феникс, 2007. – 446 с. – Доступно: http://physics-for-students.ru/bookpc/informatika/Sobol.rar