МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО
ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ
ПИ (филиал) ДГТУ в г. Таганроге

ЦМК "ПРИКЛАДНАЯ ИНФОРМАТИКА"

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

по дисциплине ОП.10 «Основы искусственного интеллекта»

Таганрог

2026

Составители: М.С. Полищук

Методические рекомендации по выполнению практических работ по дисциплине ОП.10 «Основы искусственного интеллекта» ПИ (филиал) ДГТУ в г. Таганроге, 2026 г.

В методических рекомендациях кратко изложены теоретические вопросы, необходимые для успешного выполнения практических работ, рабочее задание и контрольные вопросы для самопроверки.

Предназначено для обучающихся по направлению подготовки специальности 09.02.08 «Интеллектуальные интегрированные системы»

Ответственный за выпуск:

Председатель ЦМК: О.В. Андриян
Ф.И.О.

Издательский центр ДГТУ, 2026 г

Введение

В учебно-методических указаниях к практикуму по курсу «Основы искусственного интеллекта» изложены сведения, необходимые для успешного выполнения практических занятий по данному курсу. Описан процесс работы с инструментарием, применяемым в практических работах, представлен ряд типичных задач и подходы к их решению. Практические занятия посвящены знакомству обучающихся с основными технологиями проектирования и разработки систем искусственного интеллекта. Цель настоящего пособия – помочь обучающимся при выполнении практических работ, выполняемых для закрепления знаний по теоретическим основам и получения практических навыков работы на компьютерах

Обучающийся должен знать: методики сбора и обработки информации для решения поставленных задач с использованием технологий искусственного интеллекта, нейронных сетей, методов многомерного анализа данных.

Принципы поиска, хранения, обработки, анализа и представления информации с использованием технологий искусственного интеллекта, нейронных сетей, методов многомерного анализа данных

Методы моделирования, анализа для совершенствования бизнес-процессов и информационно-технологической инфраструктуры предприятия для достижения стратегических целей с использованием современных методов программного инструментария.

Способы проведения исследования и анализа рынка информационных систем и ИКТ, для рационального управления бизнесом.

Обучающийся должен уметь: Правильно определять стратегические цели с использованием методов современного программного инструментария.

Проводить разведочный анализ данных, проводить предобработку и очистку данных, работать с пропущенными значениями.

Проектировать базы знаний с использованием методов инженерии знаний, использовать методы анализа данных, нтерпретировать результаты анализа данных, прогнозировать поведение сложных систем. Обоснованно выбирать аиболее подходящие алгоритмы решения задач машинного обучения и оценивать качество построенных моделей.

Строить математические и компьютерные модели технических устройств и технологических процессов с использованием технологий искусственного интеллекта, нейронных сетей, методов многомерного анализа данных.

Практическое занятие № 1.

Программный инструментарий разработки систем, основанных на знаниях

Теоретическая часть

Представление знаний в системах искусственного интеллекта

Данными называют информацию фактического характера, описывающую объекты, процессы и явления предметной области, а также их свойства.

Знания – это закономерности предметной области (принципы, связи, законы), полученные в результате практической деятельности и профессионального опыта, позволяющие специалистам ставить и решать задачи в этой области. Знания описывают не только отдельные факты, но и взаимосвязи между ними, поэтому знания иногда называют структурированными данными.

База знаний – совокупность программных средств, обеспечивающих поиск, хранение, преобразование и запись в памяти ЭВМ сложно структурированных информационных единиц – знаний.

Проблема представления знаний в ИИС чрезвычайно актуальна, поскольку их функционирование опирается на знания о проблемной области, хранящиеся на компьютере. В рамках этой проблемы решаются задачи, связанные с формализацией и представлением знаний в ИИС. Для этого разрабатываются специальные модели представления знаний и языки для описания знаний, выделяются различные типы знаний.

Выделяют два вида моделей предоставления знаний: декларативные и процедурные. В ***декларативных*** моделях предметная область представляется в виде синтаксического описания еѐ состояния. Вывод решений основывается на процедурах поиска в пространстве состояний. В ***процедурном*** представлении знания содержатся в небольших программах (процедурах), которые определяют поведение ИИС. При этом можно не описывать все возможные состояния среды или объекта для реализации вывода. Достаточно хранить некоторые начальные состояния и процедуры, генерирующие необходимые описания ситуаций и действий.

К типовым декларативным моделям относят семантические сети и фреймы, а типовым процедурным моделям – исчисления высказываний / предикатов, системы продукций, нечѐткая логика. На практике редко удаѐтся обойтись рамками одной модели при разработке ИИС, поэтому представление знаний получается сложным.

Семантическая сеть представляет собой ориентированный граф, вершинами которого являются информационные единицы, имеющие индивидуальные имена. В качестве информационной единицы могут выступать события, действия, обобщѐнные понятия или свойства объектов. Вершины графа соединяются дугой, если соответствующие информационные единицы находятся в каком-либо отношении.

Фрейм представляет собой структуру данных, дающую целостное представление об объектах, явлениях и их типах в виде абстрактных образов. Структура фрейма записывается в виде списка свойств (слотов). Каждый фрейм имеет специальный слот, заполненный наименованием представляемой сущности, а другие заполнены значениями разнообразных атрибутов, ассоциирующихся с объектом.

Логика высказываний представляет собой формальную систему, элементами которой являются простые высказывания, из простых высказываний образуются сложные с помощью логических знаков

(связок). В логике изучается строение сложных высказываний, выраженных формулами, вне зависимости от содержания составляющих их простых высказываний.

Логика предикатов является расширением логики высказываний. Основным объектом здесь является переменное высказывание (предикат), истинность и ложность которого зависят от значения его переменных. Язык логики предикатов является более мощным по сравнению с языком логики высказываний. Он пригоден для формализации понятий многих проблемных областей.

Продукционная модель, или модель, основанная на правилах, позволяет представить знания в виде предложений типа ЕСЛИ (условие), ТО (действие).

Количественные данные (знания) могут быть неточными. Для учѐта неточности лингвистических знаний используется формальный аппарат ***нечѐткой алгебры***. Одно из главных понятий в нечѐткой логике – это понятие лингвистической переменной, которое определяется через нечѐткие множества. Нечѐткие множества позволяют учитывать субъективные мнения отдельных экспертов.

Обзор основных инструментов и технологий для создания систем искусственного интеллекта на основе знаний.

1. Экспертные системы:
- Экспертные системы являются одним из ключевых инструментов для работы с знаниями. Они основаны на правилах вывода, базах знаний и механизмах рассуждения. Примеры инструментов: CLIPS, Drools и RBS.
2. Логическое программирование:
- Логическое программирование широко используется для представления и обработки знаний. Одним из наиболее распространенных языков является Prolog.
3. Языки базы знаний:
- Существует множество языков для представления знаний, таких как OWL (Web Ontology Language), RDF (Resource Description Framework) и другие, которые позволяют описывать знания в семантической форме.
4. Стандартные библиотеки и фреймворки:
- Для разработки систем на базе знаний часто используются стандартные библиотеки и фреймворки, например, Apache Jena для работы с семантическими данными, Stanford CoreNLP для анализа естественного языка и другие.
5. Интегрированные среды разработки (IDE):
- Существует ряд специализированных IDE для разработки систем искусственного интеллекта на базе знаний. Например, Protege для создания и редактирования онтологий, Eclipse с плагинами для работы с логическим программированием и другие.
6. Инструменты для машинного обучения:
- Для создания систем искусственного интеллекта на базе знаний также могут использоваться инструменты для машинного обучения, такие как TensorFlow, scikit-learn, PyTorch и др.
Эти инструменты и технологии обеспечивают разработчикам широкие возможности для создания систем искусственного интеллекта на базе знаний с различными уровнями сложности и функциональности..

Цель работы: ознакомление с инструментами и технологиями для создания систем искусственного интеллекта на основе знаний..

Рабочее задание

Задание Вариант № 1 создать экспертную систему для диагностики болезней. Система должна иметь базу знаний, содержащую симптомы, диагнозы и правила логического вывода. Для реализации системы можно использовать один из инструментов, например, CLIPS или Prolog.

Шаги выполнения задания:

1. Создание базы знаний: определить симптомы, возможные диагнозы и правила для вывода диагноза на основе симптомов.

2. Реализация экспертной системы с использованием выбранного инструмента.
3. Тестирование системы: проверить работу системы, подавая на вход различные комбинации симптомов и проверяя правильность вывода диагноза.

Задание вариант № 2.
Разработать систему для автоматической рекомендации книг на основе пользовательских предпочтений. Система должна иметь базу знаний, содержащую информацию о книгах, историю оценок и предпочтений пользователей. Для реализации системы можно использовать выбранный ранее инструмент, например, CLIPS или Prolog.

Шаги выполнения задания:

1. Создание базы знаний: определить информацию о книгах, историю оценок и предпочтений пользователей.
2. Реализация системы для автоматической рекомендации книг на основе пользовательских предпочтений.
3. Тестирование системы: проверить работу системы, подавая на вход предпочтения разных пользователей и проверяя правильность рекомендаций книг.

Контрольные вопросы

1. Что такое экспертная система, и какие основные компоненты включает в себя такая система?
2. Какие инструменты и технологии можно использовать для создания экспертных систем на базе знаний?
3. Какая роль базы знаний в экспертной системе, и какие типы знаний могут быть представлены в базе?
4. Каким образом осуществляется логический вывод в экспертных системах, и какие алгоритмы используются для этого?
5. Какие этапы включает процесс разработки экспертной системы для диагностики болезней?
6. Каким образом можно провести тестирование и отладку экспертной системы для уверенности в ее корректной работе?
7. Какие преимущества и недостатки свойственны экспертным системам по сравнению с традиционными программами?
8. Какие методы логического программирования можно применить для реализации экспертных систем?
9. Какие возможности для расширения и модернизации экспертных систем существуют на сегодняшний день?
10. Каким способом можно управлять базой знаний экспертной системы, чтобы обеспечить ее актуальность и эффективность?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета: титульный лист;

- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 2.

Модели представления данных и знаний

Теоретическая часть

Для построения модели представления знаний в виде графа необходимо выполнить следующие шаги:

1) Определить целевые действия задачи (являющиеся решениями).

2) Определить промежуточные действия или цепочку действий, между начальным состоянием и конечным (между тем, что имеется, и целевым действием).

3) Опередить условия для каждого действия, при котором его целесообразно и возможно выполнить. Определить порядок выполнения действий.

4) Добавить конкретные факты , исходя из поставленной задачи.

5) Преобразовать полученный порядок действий и соответствующие им факты, условия и действия.

6) Для проверки правильности построения записать цепочки, явно проследив связи между ними. Этот набор шагов предполагает движение при построении модели от результата к начальному состоянию, но возможно и движение от начального состояния к результату (шаги 1 и 2).

7) Присвоить обозначения фактам Ф, правилам П, действиям Д.

8) Построить граф предметной области. (пример рис.1)

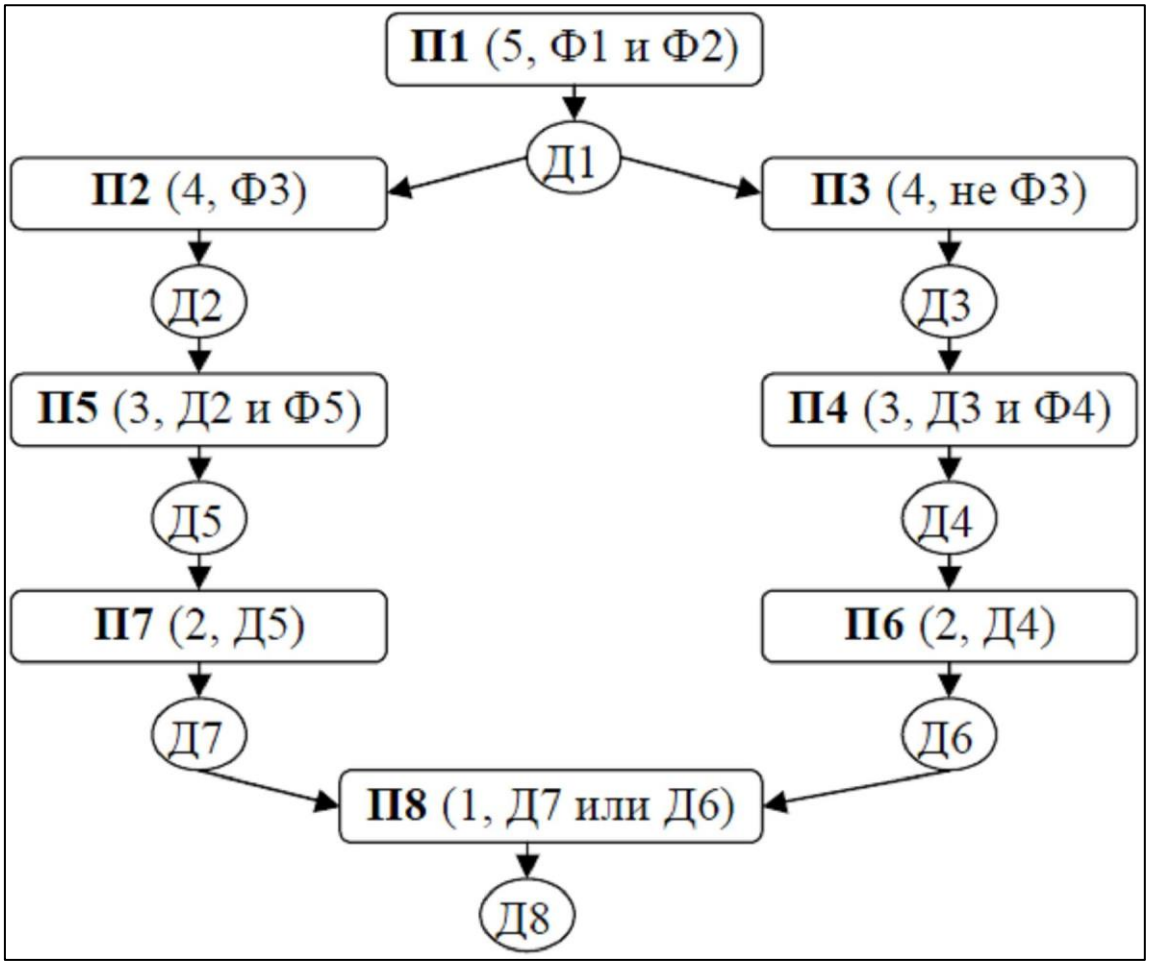


Рис. 1 – Пример графа модели знаний.

Цель работы: Изучить заданную предметную область и построить модель знаний в виде графа.

Рабочее задание

1. Изучить теоретическую часть по приведенным выше данным и дополнительной литературе;
2. Просмотреть демонстрационный пример;
3. Получить у преподавателя вариант задания для выполнения;
4. Построить графовую модель заданного объекта;
5. Реализовать программу с использованием графовой модели
6. Оформите отчет.

Варианты заданий

1.Построить модель представления знаний в предметной области «Железная дорога» (продажа билетов).

2.Построить модель представления знаний в предметной области «Торговый центр» (организация).

3.Построитьмодельпредставлениязнанийвпредметнойобласти «Автозаправка» (обслуживание клиентов).

4.Построитьмодельпредставлениязнанийвпредметнойобласти «Компьютерные сети» (организация).

5.Построитьмодельпредставлениязнанийвпредметнойобласти «Университет» (учебный процесс).

6.Построитьмодельпредставлениязнанийвпредметнойобласти «Компьютерная безопасность» (средства и способы ее обеспечения).

7.Построитьмодельпредставлениязнанийвпредметнойобласти «Компьютерная безопасность» (угрозы).

8.Построить модель представления знаний в предметной области «Интернет- кафе» (организация и обслуживание).

9.Построитьмодельпредставлениязнанийвпредметнойобласти «Разработка информационных систем» (ведение информационного проекта).

10.Построитьмодельпредставлениязнанийвпредметнойобласти «Туристическое агентство» (работа с клиентами).

11.Построить модель представления знаний в предметной области «Кухня» (приготовление пищи).

12.Построитьмодельпредставлениязнанийвпредметнойобласти «Больница» (прием больных).

13.Построитьмодельпредставлениязнанийвпредметнойобласти «Кинопрокат» (ассортимент и работа с клиентами).

14.Построить модель представления знаний в предметной области «Прокат автомобилей» (ассортимент и работа с клиентами).

15.Построитьмодельпредставлениязнанийвпредметнойобласти «Операционные системы» (функционирование).

16.Построитьмодельпредставлениязнанийвпредметнойобласти

17.Построить модель представления знаний в предметной области «Предприятие» (структура и функционирование).

Контрольные вопросы

1.Какие преимущества предлагает модель знаний в виде графа по сравнению с другими структурами данных для представления информации?

2.Какие основные типы графовых моделей знаний существуют, и как они могут быть применены в различных областях, таких как информационный поиск, обработка естественного языка и машинное обучение?

3.Какие методы и алгоритмы используются для анализа и извлечения знаний из графовых моделей, и как они помогают в решении различных задач?

4.Какие вызовы и проблемы возникают при построении и обновлении моделей знаний в виде графа, и какие подходы предлагаются для их решения?

5.Каким образом модели знаний в виде графа могут использоваться для семантического анализа текста, распознавания образов и других задач, требующих представления и обработки знаний?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа

Практическое занятие № 3.

Программный инструментарий разработки систем, основанных на знаниях

Теоретическая часть:

Общие сведения о пакете Fuzzy Logic Toolbox

Для рассмотрения результатов разработки и функционирования систем нечёткой логики будем использовать графические средства пакета **Fuzzy Logic Toolbox**. Эти же средства используются и при разработке систем нечёткого вывода как графический объектно-ориентированный язык автоматического программирования.

В состав этих средств входят:

— редактор систем нечёткого вывода **FIS Editor** (FIS);

— редактор функций принадлежности систем нечёткого вывода **Membership Function Editor** (MFE);

— редактор правил систем нечёткого вывода **Rule Editor**;

— программа просмотра правил системы нечёткого вывода **Rule Viewer**;

программа просмотра поверхности нечёткого вывода **Sur-face Viewer**. Дляописаниянечётких высказыванийиспользуютсянечёткие лингвистическиепеременные(ЛП). ЛП — это именованная переменная, которая принимает свои значения из множествалингвистическихтермов,т.е.символьных величин. Для нечёткой ЛП терм-множество задаётся как нечёткое множество. Этот процессназывается фаззификацией. Фаззификация является одной из проблемных задач описания нечёткого вывода и отражает индивидуальные эмпирические знания автора. Нечёткие высказывания в условной части нечёткой продукции могут быть составными, соединёнными связками “И” и/или “ИЛИ”. Эти связки при исчислении высказываний реализуются логическими или арифметическими операциями пересечения или объединения, соответственно.

При получении результата по каждому правилу необходимо дать оценку степени его истинности. Эта оценка зависит от степени истинности высказываний условной части правила, степени истинности отношения, положенного в основу правила, между исходными утверждениями (посылкой) и заключением, т.е. сте-пени истинности импликации, и степени истинности высказыва-ния относительно значения из терм- множества возможных ре-зультатов, приведенного в правиле. Получение оценки степени истинности заключения, полученного по правилу,называютактивизацией. В случае необходимости получения чёткого количественно-го значения результата оно может быть получено на основании функции принадлежности терм-множества результата различными способами по алгоритмам, названым по именам их авторов (Мамдани, Сугено, Цукамото и т.д.), что определяет тип системы нечёткого вывода. Эта операция называется дефаззификацией.

Редактор функций принадлежности (MFE)

Редактор функций принадлежности в графическом режиме обеспечивает задание и изменение функции принадлежности любых термов ЛП СНВ.

Для фаззификации лингвистической переменной СНВ следует выделить ее изображение – именованный прямоугольник в левой верхней части окна редактора (см. рис. 1).

В окне редактора выводятся графики функций принадлежности для всех значений выделенной ЛП (по умолчанию для трёх значений).

Для описания функции принадлежности каждого значения ЛП используются три поля: **Name**, **Type** и **Params**. Описываемая функция выделяется щелчком по её графику. В поле **Name** устанавливается значение ЛП. В поле **Type**, выбором элемента меню, устанавливается имя нужной функции

принадлежности (одной из 11-ти встроенных). В поле ввода **Params** указываются необходимые параметры функции принадлежности, которые определяют положение ее модальных значений на числовой шкале, диапазон изменения которой указывается в полях ввода **Range** и **Display range**.

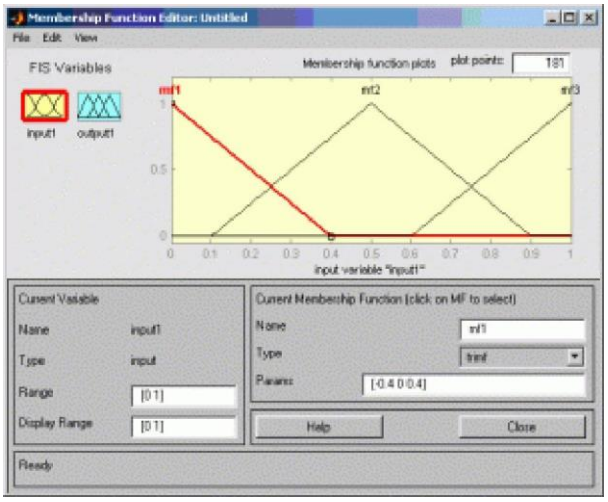


Рис. 1. Окно редактора Membership Function Editor

Эти операции выполняются над всеми значениями из терм-множеств лингвистических переменных СНВ.

Добавление нового значения ЛП со встроенной функцией принадлежности производится по команде основного меню **Edit > Add MF**.

Удаление ненужного значения ЛП производится нажатием клавиши **Delete**, после выделения графика функции принадлежности этого значения.

Цель работы: ознакомится со способами и средствами описания нечётких множеств и продукций в системе нечёткого вывода в интерактивном режиме использования графических средств пакета **Fuzzy Logic Toolbox**.

Рабочее задание

Задание. Для полученного варианта и разработанного графа ПрО сформировать лингвистические переменные и получить для них функции принадлежности.

Контрольные вопросы:

1. Что такое нечёткое множество и какие компоненты определяют его характеристики?
2. Каким образом можно представить нечёткое правило в системе нечёткого вывода, используя продукцию?
3. Какие графические средства в пакете Fuzzy Logic Toolbox позволяют визуализировать нечёткие множества?
4. Какие шаги необходимо выполнить для построения и анализа модели на основе нечёткой продукции в MATLAB?
5. В чём заключается разница между традиционными и нечёткими системами, и какие преимущества обладают нечеткие системы в конкретной задаче решения?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа

Практическое занятие № 4.

Модели представления данных и знаний

Теоретическая часть

Фреймы - один из распространенных формализмов представления знаний в ЭС. Фрейм можно представить себе как структуру, состоящую из набора ячеек - слотов. Каждый слот состоит из имени и ассоциируемых с ним значений. Значения могут представлять собой данные, процедуры, ссылки на другие фреймы или быть пустыми. Такое построение оказывается очень удобным для моделирования аналогий, описания областей с родовидовыми связями понятий и т.п.

Любой фрейм состоит из некоторых составляющих, имена и содержание которых описано ниже:

1. Имя фрейма. Это идентификатор, присваиваемый фрейму, фрейм должен иметь имя уникальное в данной фреймовой системе.
2. Имя слота. Это идентификатор, присваиваемый слоту; слот должен иметь уникальное имя во фрейме, к которому он принадлежит. Обычно имя слота не несет никакой смысловой нагрузки и является лишь идентификатором данного слота.

Указатели наследования. Эти указатели касаются только фреймовых систем иерархического типа, основанные на отношениях “абстрактное-конкретное”, они

показывают, какую информацию об атрибутах слотов во фрейме верхнего уровня наследуют слоты с такими же именами во фрейме нижнего уровня. Типичные указатели наследования Unique (U: - уникальный), Same (S: такой же), Range (R: установление границ), Override (O: игнорировать) и т.п. U показывает, что фрейм может иметь слоты с разными значениями: S - все слоты должны иметь одинаковые значения, R - значение слотов фрейма нижнего уровня должны находиться в пределах, указанных значениями слотов фрейма верхнего уровня, O - при отсутствии указания значение слота фрейма верхнего уровня становится значением слота фрейма нижнего уровня, но в случае определения нового значения слотов фреймов нижних уровней указываются в качестве значений слотов.

3. Указание типа данных. указывается, что слот имеет численное значение, либо служит указателем другого фрейма. К типам данных относятся:
   FRAME (указатель), INTEGER (целый), REAL (действительный), BOOL (булев), LISP (присоединенная процедура), TEXT (текст), LIST (список), TABLE (таблица), EXPRESSION (выражение) и др.

4. Значение слота. Пункт ввода значения слота. Значение слота должно совпадать с указанным типом данных этого слота, кроме того должно выполняться условие наследования.
5. Демон. Здесь дается определение демонов типа IF-NEEDED, IF-ADDED, IF-REMOVED и т.д. Демоном называется процедура, автоматически запускаемая при выполнении некоторого условия. демоны запускаются при обращении к соответствующему слоту. Кроме того, демон является разновидностью присоединенной процедуры.
6. Присоединенная процедура. В качестве значения слота можно использовать программу процедурного типа. Когда мы говорим, что в моделях представления знаний фреймами объединяются процедурные и декларативные знания, то считаем демоны и присоединенные процедуры процедурными знаниями.

Особенностью иерархической структуры является то, что информация об атрибутах фрейма на верхнем уровне совместно используется всеми фреймами нижних уровней, связанных с ним.

Например: Фреймовое представление конференции.

Иерархические фреймовые структуры базируются на отношениях IS – A между фреймами, описывающими некоторую конференцию. Все фреймы должны содержать информацию о ДАТЕ, МЕСТЕ, НАЗВАНИИ ТЕМЫ, ДОКЛАДЧИКЕ. Таким образом, на самом верхнем уровне определен фрейм КОНФЕРЕНЦИЯ.

Конференции разделяются на коммерческие и по развитию. Они составляют дочерние фреймы. В них могут быть добавлены слоты: объем торговли и бюджет.

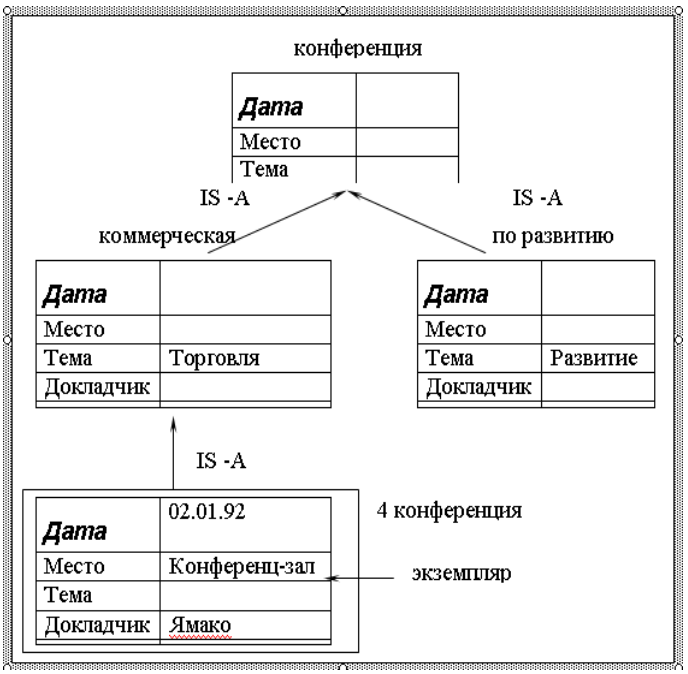


Рис.3. Пример фреймовой модели

Цель работы: Научиться использовать фреймы для представления знаний в интеллектуальных системах

Рабочее задание:

1. Изучить теоретическую часть по приведенным выше данным и дополнительной литературе.
2. Просмотреть демонстрационный пример.
3. Получить у преподавателя вариант задания для выполнения.
4. Построить фреймовую модель заданного объекта;
5. Реализовать программу с использованием фреймовой модели

Варианты заданий

Используя фреймовую модель представления знаний реализовать структуру отношений, описывающие следующие ситуации:

1. экзамен по дисциплине за семестр у преподавателя при составляющих: семестр, экзамен, преподаватель, оценка, студент, получать.

2. ведомость при составляющих: дисциплина, студент, экзамен, семестр, преподаватель, оценка.
3. конференция по коммерческим вопросам при составляющих: дата, место проведения, тема, цель выступающие.
4. получение оценки при составляющих: преподаватель, студент, оценка, получать.
5. использования изделия при составляющих: организация, разработка технологического решения, исследование «физического эффекта», методы создания изделия.
6. информационная структура БД в машиностроении при составляющих: физические эффекты, технические решения, изделия, объект поставки изделия, приборы и стенды, нормативы.
7. классификация продукта при составляющих: название, область применения, способ хранения, способ транспортировки.
8. аудитория (описание) при составляющих: вместимость, назначение, составляющие, местонахождение.
9. животный мир при составляющих: вид, тип, среда обитания, особенности поведения.

Контрольные вопросы

7. Что представляет из себя фрейм, его составные части?
8. Что такое слот и из каких частей он состоит?
9. Для чего служат имя фрейма и имя слота?
10. Для чего служат указатели наследования?
11. для чего служат указание типа данных, демон?
12. Для чего служат присоединенная процедура и значение слота?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа

Практическое занятие № 5.

Программная реализация алгоритма обратного распространения ошибки.

Теоретическая часть

Формальные персептроны (или, лучше говорить - нейроны) могут объединяться в сети различными способами. Рассмотрим простейшее объединение – многослойный персептрон прямого распространения сигнала (связи соотвественно идут направлено от i-го слоя к i+1 - му). Различают:

Входной слой - это начальный уровень сети, который принимает входные данные, которые будут использоваться для создания выходных данных. Входной слой не производит никаких вычислений – просто передает сигнал на первый скрытый слой (в нейронах входного слоя нет функций активации).

Скрытые слои (может быть не только один скрытый слой) - в сети должен быть хотя бы один скрытый слой. Скрытые слои выполняют вычисления и операции с входными данными, чтобы получить результат на выходе.

Скрытые слои связаны между собой синаптическими связями – каждый нейрон предыдущего с слоя с каждым нейроном последующего слоя.

Выходной слой – собственно выдает результат вычисления сети. В нейронах выходного слоя также есть функции активации.

Обычно функции активации одни и те же в нейронах скрытых слоев и выходном слое. Но могут быть и исключения (в данном случае нужен «хитрый» способ обучения). В то время как входные нейроны берут свои значения из окружения, значения всех других нейронов вычисляются с помощью математической функции, включающей веса и значения предшествующего слоя

Каждый слой рассчитывает нелинейное преобразование от линейной комбинации сигналов предыдущего слоя.

Многослойная нейронная сеть может формировать на выходе произвольную многомерную функцию при соответствующем выборе количества слоев, диапазона изменения сигналов и параметров нейронов.

Парадигмы обучения нейронных сетей

Обычно выделяют две вида обучения нейронных сетей:

- с учителем – есть и входы и выходы;

- без учителя – есть только входы.

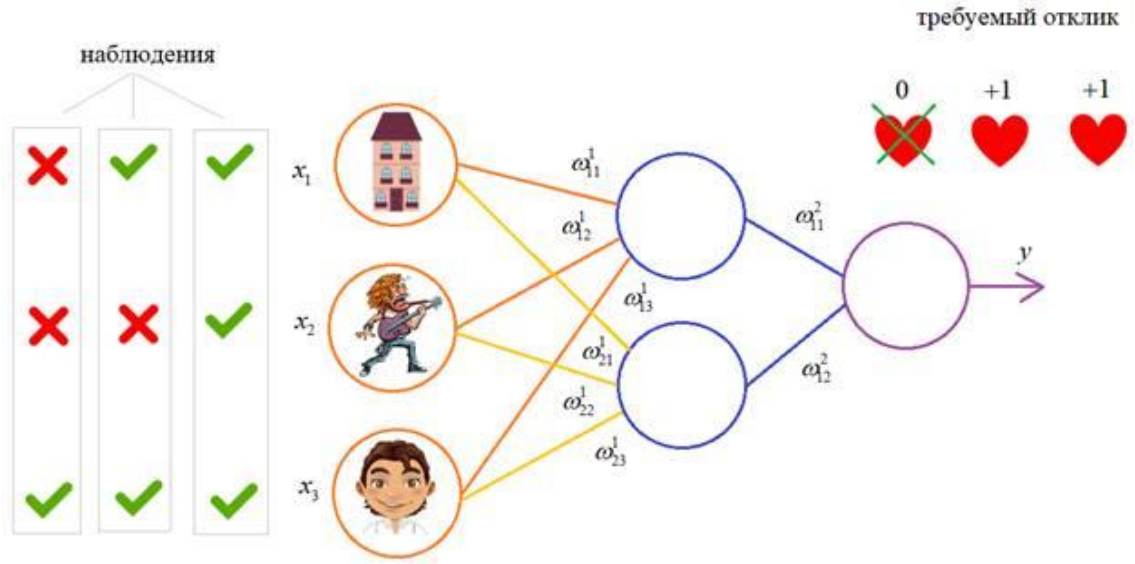
Но, есть еще промежуточная форма – обучение с подкреплением

Обобщенный алгоритм обучения нейронных сетей с учителем следующий:

- Шаг 1. Подготовить обучающую выборку (входы - выходы);
- Шаг 2. Предварительно рассчитать структуру сети;
- Шаг 3. Инициализировать случайным образом матрицы весов синаптических связей нейронной сети;
- Шаг 4. Подать выбранный случайным образом пример из обучающей выборки и рассчитать выходы сети (прямой проход);
- Шаг 5. Рассчитать ошибку на выходе сети и с помощью специальных формул скорретировать веса синаптических связей. Если ошибка сети меньше заданной, то вернуться на шаг 3.
- (Шаг 6) Проверить на валидационной выборке, если ошибка удовлетворяет разработчика, то сеть – в реальный «боевой» режим;
- (Шаг 7) Оптимизация работы сети – сокращение числа слоев и нейронов; 5
- (Шаг 8). В режиме эксплуатации зачатую необходимо переобучение, т.к. появляются новые данные. В скобках указаны не обязательные, но на практике необходимые шаги.

Алгоритм обратного распространения ошибки
Алгоритм обратного распространения ошибки был опубликован в 1986 году в работе Руммельхарта, Хинтона и Вильямса, которые считаются первооткрывателями применения градиентного спуска при обучении нейронных сетей, но первым данный алгоритм предложил Вербос в своей кандидатской диссертации в 1974 году, а также наш, советский кибернетик Галушкин в том же году.
Искусственные нейронные сети используют обратное распространение в качестве алгоритма обучения для вычисления градиентного спуска с учетом весов
Желаемые выходы сравниваются с достигнутыми выходами системы, а затем системы настраиваются путем регулировки веса соединений, чтобы максимально сократить разницу между ними.
Алгоритм получил свое название, в связи с тем, что веса обновляются в обратном направлении, от вывода к вводу. Т.е. нам известна ошибка на выходе, далее считаем изменение весов на последнем слое, затем, через ошибку на последнем слое, пересчитываем матрицу перед предпоследним слоем и т.д

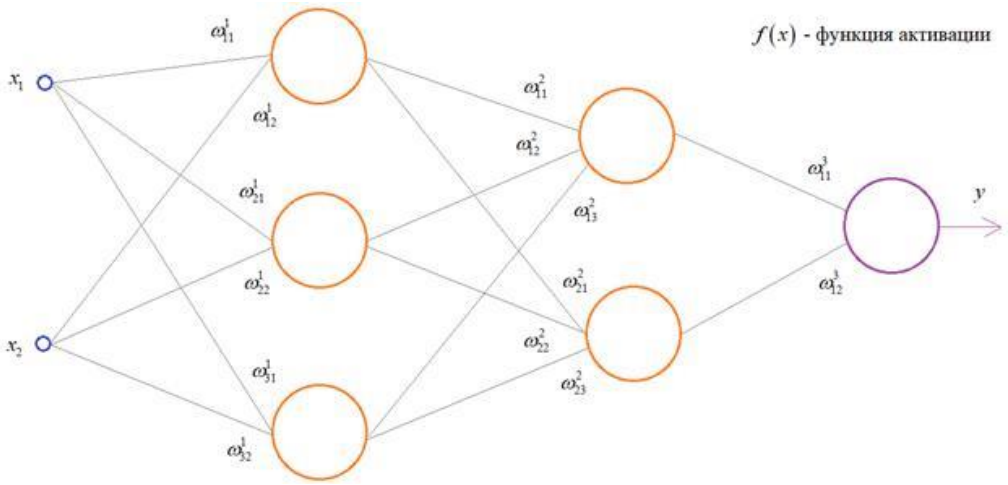
Пример реализации многослойного персептрона с алгоритмом обратного распространения ошибки для задачи глубокого обучения
Один из распространенных подходов к обучению заключается в последовательном предъявлении нейронной сети(НС) векторов наблюдений и последующей корректировки весовых коэффициентов так, чтобы выходное значение совпадало с требуемым



Это называется **обучение с учителем**, так как для каждого вектора мы знаем нужный ответ и именно его требуем от нашей НС.

Теперь, главный вопрос: как построить алгоритм, который бы наилучшим образом находил весовые коэффициенты. Наилучший – это значит, максимально быстро и с максимально близкими выходными значениями для требуемых откликов. В общем случае эта задача не решена. Нет универсального алгоритма обучения. Поэтому, лучшее, что мы можем сделать – это выбрать тот алгоритм, который хорошо себя зарекомендовал в прошлом. Основной «рабочей лошадкой» здесь является алгоритм **back propagation** (обратного распространения ошибки), который, в свою очередь, базируется на алгоритме **градиентного спуска**.

Чтобы все лучше понять, предположим, что у нас имеется вот такая полносвязная НС прямого распространения с весами связей, выбранными произвольным образом в диапазоне от [-0.5; 0,5]. Здесь верхний индекс показывает принадлежность к тому или иному слою сети. Также, каждый нейрон имеет некоторую активационную функцию $f(x)$:



На первом шаге делается прямой проход по сети. Мы пропускаем вектор наблюдения $[x_1, x_2]$ через эту сеть, и запоминаем все выходные значения нейронов скрытых слоев:

$$f_{11} = f\left(w^1_{11}x_1 + w^1_{12}x_2\right)$$
$$f_{12} = f\left(w^1_{21}x_1 + w^1_{22}x_2\right) \quad f_{21} = f\left(w^2_{11}f_{11} + w^2_{12}f_{12} + w^2_{13}f_{13}\right)$$
$$f_{13} = f\left(w^1_{31}x_1 + w^1_{32}x_2\right) \quad f_{22} = f\left(w^2_{21}f_{11} + w^2_{22}f_{12} + w^2_{23}f_{13}\right)$$

и последнее выходное значение y:

$$v_{out} = w^3_{11} \cdot f_{21} + w^3_{12} \cdot f_{22}$$
$$y = f\left(v_{out}\right)$$

Далее, мы знаем требуемый отклик d для текущего вектора $[x_1, x_2]$, значит для него можно вычислить ошибку работы НС. Она будет равна:

$$e = y - d$$

А вот дальше начинается самое главное – корректировка весов. Для этого делается обратный проход по НС: от последнего слоя – к первому.

Итак, у нас есть ошибка e и некая функция активации нейронов $f(x)$. Первое, что нам нужно – это вычислить локальный градиент для выходного нейрона. Это делается по формуле:

$$\delta = e \cdot f'(v_{out})$$

Этот момент требует пояснения. Смотрите, ранее используемая пороговая функция:

$$f(x) = \begin{cases} 1, & x \geq 0,5 \\ 0, & x < 0,5 \end{cases}$$

нам уже не подходит, т.к. она не дифференцируема на всем диапазоне значений x. Вместо этого для сетей с небольшим числом слоев, часто применяют
или **гиперболический тангенс**:

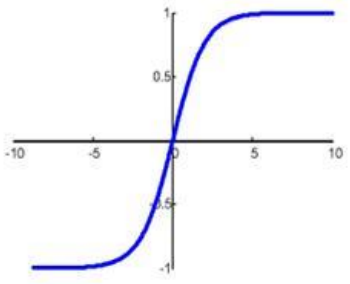
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

или **логистическую функцию**:

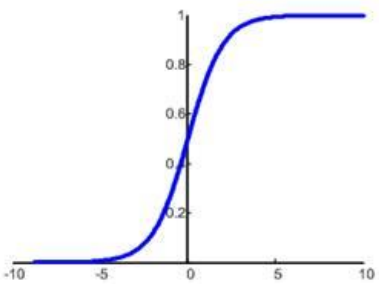
$$f(x) = \frac{1}{1 + e^x}$$

Фактически, они отличаются только тем, что первая дает выходной интервал [-1; 1], а вторая – [0; 1]. И мы уже берем ту, которая нас больше устраивает в данной конкретной ситуации. Например, выберем логистическую функцию.

гиперболический тангенс



логистическая функция



Ее производная функции по аргументу x дает очень простое выражение:

$$f'(x) = f(x) \cdot (1 - f(x))$$

Именно его мы и запишем в нашу формулу вычисления локального градиента:

$$\delta = e \cdot f'(v_{out}) = e \cdot f(v_{out}) \cdot (1 - f(v_{out}))$$

Но, так как

$$y = f\left(v_{out}\right)$$

то локальный градиент последнего нейрона, равен:

$$\delta = e \cdot y \cdot (1 - y)$$

Отлично, это сделали. Теперь у нас есть все, чтобы выполнить коррекцию весов.

Начнем со связи w_{11}^3, формула будет такой:

$$w_{11}^3 = w_{11}^3 - \begin{bmatrix} \lambda \\ шаг \\ сходимости \end{bmatrix} \cdot \begin{bmatrix} \delta \\ локальный \\ градиент \end{bmatrix} \cdot \begin{bmatrix} f_{21} \\ выходной \\ сигнал \end{bmatrix}$$

Для второй связи все то же самое, только входной сигнал берется от второго нейрона:

$$w_{12}^3 = w_{12}^3 - \begin{bmatrix} \lambda \\ шаг \\ сходимости \end{bmatrix} \cdot \begin{bmatrix} \delta \\ локальный \\ градиент \end{bmatrix} \cdot \begin{bmatrix} f_{22} \\ выходной \\ сигнал \end{bmatrix}$$

Здесь у вас может возникнуть вопрос: что такое параметр λ и где его брать? Он подбирается самостоятельно, вручную самим разработчиком. В самом простом случае можно попробовать следующие значения:

$$\lambda = 0,1;\ 0,01;\ 0,001$$

Итак, мы с вами скорректировали связи последнего слоя. Если вам все это понятно, значит, вы уже практически поняли весь алгоритм обучения, потому что дальше действуем подобным образом. Переходим к нейрону следующего с конца слоя и для его входящих связей повторим ту же саму процедуру. Но для этого, нужно знать значение его локального градиента. Определяется он просто. Локальный градиент последнего нейрона взвешивается весами входящих в него связей. Полученные значения на каждом нейроне умножаются на производную функции активации, взятую в точках входной суммы:

$$\delta_{21} = \delta \cdot w_{11}^3 \cdot f'\left(v_{21}\right) = \delta \cdot w_{11}^3 \cdot \left[f_{21} \cdot (1 - f_{21})\right]$$

$$\delta_{22} = \delta \cdot w_{12}^3 \cdot f'\left(v_{22}\right) = \delta \cdot w_{12}^3 \cdot \left[f_{22} \cdot (1 - f_{22})\right]$$

А дальше действуем по такой же самой схеме, корректируем входные связи по той же формуле:

$$w_{11}^2 = w_{11}^2 - \lambda \cdot \delta_{21} \cdot f_{11}$$

$$w_{12}^2 = w_{12}^2 - \lambda \cdot \delta_{21} \cdot f_{12}$$

$$w_{13}^2 = w_{13}^2 - \lambda \cdot \delta_{21} \cdot f_{13}$$

И для второго нейрона:

$$w_{21}^2 = w_{21}^2 - \lambda \cdot \delta_{22} \cdot f_{11}$$

$$w_{22}^2 = w_{22}^2 - \lambda \cdot \delta_{22} \cdot f_{12}$$

$$w_{23}^2 = w_{23}^2 - \lambda \cdot \delta_{22} \cdot f_{13}$$

Осталось скорректировать веса первого слоя. Снова вычисляем локальные градиенты для нейронов первого слоя, но так как каждый из них имеет два выхода, то сначала вычисляем сумму от каждого выхода:

$$\sigma_{11} = \delta_{21} \cdot w_{11}^2 + \delta_{22} \cdot w_{21}^2$$
$$\sigma_{12} = \delta_{21} \cdot w_{12}^2 + \delta_{22} \cdot w_{22}^2$$
$$\sigma_{13} = \delta_{21} \cdot w_{13}^2 + \delta_{22} \cdot w_{23}^2$$

А затем, значения локальных градиентов на нейронах первого скрытого слоя:

$$\delta_{11} = \sigma_{11} \cdot f'(v_{11}) = \sigma_{11} \cdot \left[f_{11} \cdot (1 - f_{11}) \right]$$
$$\delta_{12} = \sigma_{12} \cdot f'(v_{12}) = \sigma_{12} \cdot \left[f_{12} \cdot (1 - f_{12}) \right]$$
$$\delta_{13} = \sigma_{13} \cdot f'(v_{13}) = \sigma_{13} \cdot \left[f_{13} \cdot (1 - f_{13}) \right]$$

Ну и осталось выполнить коррекцию весов первого слоя все по той же формуле:

$$w_{11}^1 = w_{11}^1 - \lambda \cdot \delta_{11} \cdot x_1$$
$$w_{12}^1 = w_{12}^1 - \lambda \cdot \delta_{11} \cdot x_2$$
$$w_{21}^1 = w_{21}^1 - \lambda \cdot \delta_{12} \cdot x_1$$
$$w_{22}^1 = w_{22}^1 - \lambda \cdot \delta_{12} \cdot x_2$$
$$w_{31}^1 = w_{31}^1 - \lambda \cdot \delta_{13} \cdot x_1$$
$$w_{32}^1 = w_{32}^1 - \lambda \cdot \delta_{13} \cdot x_2$$

В результате, мы выполнили одну итерацию алгоритма обучения НС. На следующей итерации мы должны взять другой входной вектор из нашего обучающего множества. Лучше всего это сделать случайным образом, чтобы не формировались возможные ложные закономерности в последовательности данных при обучении НС. Повторяя много раз этот процесс, весовые связи будут все точнее описывать обучающую выборку.

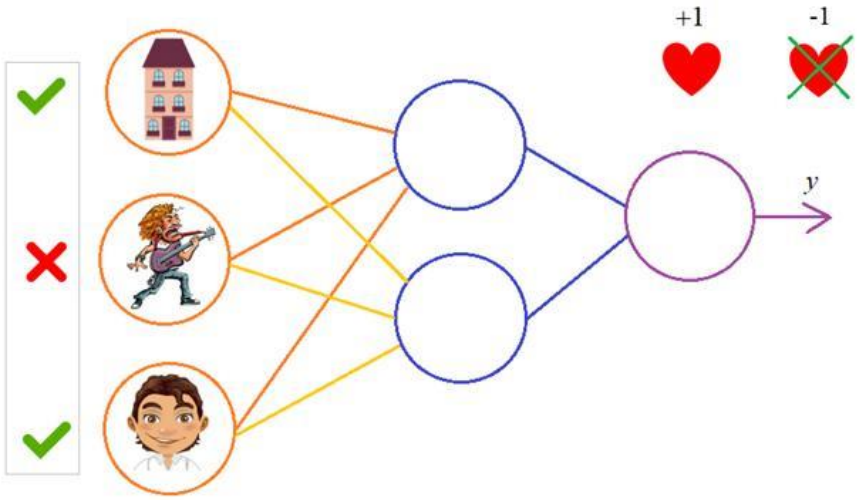
Отлично, процесс обучения в целом мы рассмотрели. Но какой критерий качества минимизировался алгоритмом градиентного спуска? В действительности, мы стремились получить минимум суммы квадратов ошибок для обучающей выборки:

$$E = \frac{1}{2} \sum_{j=1}^{N} e_j^2 = \frac{1}{2} \sum_{j=1}^{N} \left(d_j - y_j \right)^2$$

То есть, с помощью алгоритма градиентного спуска веса корректируются так, чтобы минимизировать этот критерий качества работы НС. Позже мы еще увидим, что на практике используется не только такой, но и другие критерии.

Вот так, в целом выглядит идея работы алгоритма обучения по методу обратного распространения ошибки.

Давайте теперь в качестве примера обучим следующую НС:



В качестве обучающего множества выберем все возможные варианты (здесь 1 – это да, -1 – это нет):

Вектор наблюдений	Требуемый отклик
[-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

На каждой итерации работы алгоритма, мы будем подавать случайно выбранный вектор и корректировать веса, чтобы приблизиться к значению требуемого отклика. В качестве активационной функции выберем гиперболический тангенс:

$$f(x) = \frac{2}{1+e^{-x}} - 1$$

со значением производной:

$$f'(x) = \frac{1}{2} \cdot \left(1 + f(x)\right) \cdot \left(1 - f(x)\right)$$

Программа на Python будет такой:

```
import numpy as np

def f(x):
    return 2/(1 + np.exp(-x)) - 1

def df(x):
    return 0.5*(1 + x)*(1 - x)

W1 = np.array([[-0.2, 0.3, -0.4], [0.1, -0.3, -0.4]])
W2 = np.array([0.2, 0.3])

def go_forward(inp):
    sum = np.dot(W1, inp)
```

```
    out = np.array([f(x) for x in sum])

    sum = np.dot(W2, out)
    y = f(sum)
    return (y, out)

def train(epoch):
    global W2, W1
    lmd = 0.01          # шаг обучения
    N = 10000           # число итераций при обучении
    count = len(epoch)
    for k in range(N):
        x = epoch[np.random.randint(0, count)]  # случайных выбор входного
сигнала из обучающей выборки
        y, out = go_forward(x[0:3])            # прямой проход по НС и
вычисление выходных значений нейронов
        e = y - x[-1]                          # ошибка
        delta = e*df(y)                        # локальный градиент
        W2[0] = W2[0] - lmd * delta * out[0]    # корректировка веса первой
связи
        W2[1] = W2[1] - lmd * delta * out[1]    # корректировка веса второй
связи

        delta2 = W2*delta*df(out)              # вектор из 2-х величин локальных
градиентов

        # корректировка связей первого слоя
        W1[0, :] = W1[0, :] - np.array(x[0:3]) * delta2[0] * lmd
        W1[1, :] = W1[1, :] - np.array(x[0:3]) * delta2[1] * lmd

# обучающая выборка (она же полная выборка)
epoch = [(-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)]

train(epoch)        # запуск обучения сети

# проверка полученных результатов
for x in epoch:
    y, out = go_forward(x[0:3])
```

```
print(f"Выходное значение НС: {y} => {x[-1]}")
```

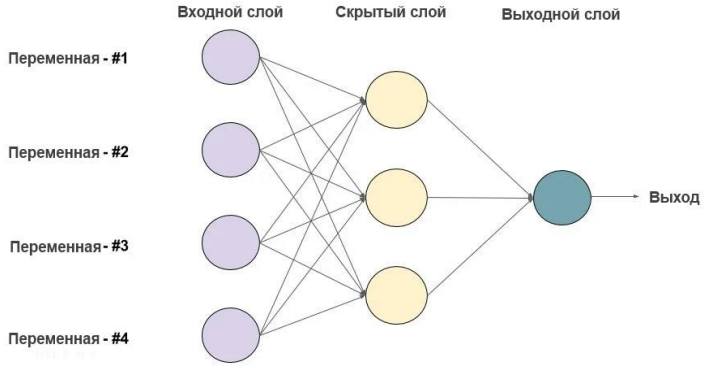
Цель работы: Реализация многослойного персептрона с алгоритмом обратного распространения ошибки для задачи глубокого обучения

Порядок выполнения работы

1.Создайте новый блакнот в среде Google Colab
2. Скопируйте в ячейку блокнота приведенный выше код
3.Запустите ячейку с кодом и получите результат работы программы
4.Измените код программы таким образом, чтобы выполнить те же вычисления для НС с другой архитектурой (см. Рабочее задание)
5. Получите результат вычислений
6.Оформите отчет и сдайте его преподавателю

Рабочее задание

Написать программу для обучения НС следующей архитектуры



Контрольные вопросы

1. Как выбирается количество слоев и нейронов в них в многослойном персептроне в зависимости от задачи и исходных данных?
2. Какие методы повышения скорости обучения многослойного персептрона Вы знаете?
3. Как влияет на скорость обучения динамическое добавление нейронов в процессе обучения?
4. Возможно ли менять структуру многослойного персептрона в процессе обучения? Если да, то как и к чему это может привести?
5. Как измеряется ошибка обучения многослойного персептрона?
6. Как определить необходимый размер обучающей выборки?
7. Как определить размер валидационной выборки?
8. Какие параметры регулируются в алгоритме обратного распространения ошибки?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа

Практическое занятие № 6.

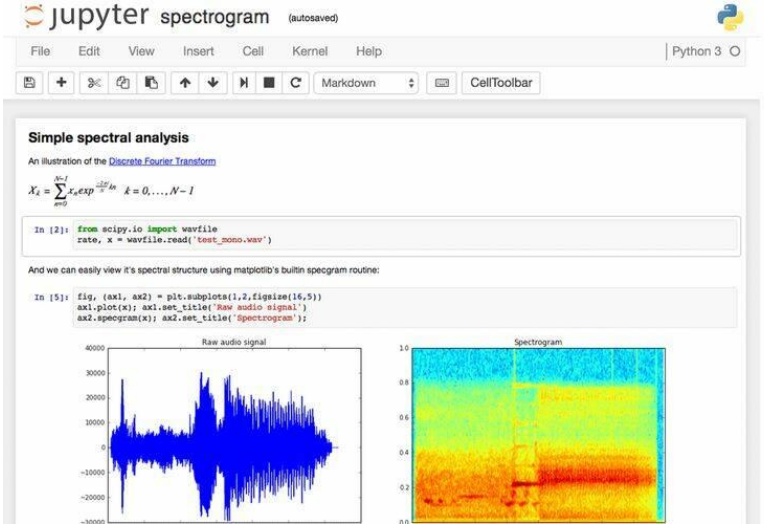
Настройка и конфигурирование программного обеспечения Jupiter.

Теоретическая часть

Jupyter notebook — веб-приложение с открытым исходным кодом. Его используют в Data Science для визуализации данных

Что такое Jupyter-ноутбук и где применяется

В мире Data Science Jupyter-ноутбук уже несколько лет считается одним из популярных инструментов для анализа данных и быстрого прототипирования.



С Jupyter-ноутбуком аналитические отчёты получаются нагляднее: можно выводить вместе код, изображения, формулы, диаграммы и графики

В Jupyter Notebook две основные части: веб-приложение и ноутбуки — файлы, в которых работают с исходным кодом программы, запускают его и выводят данные в разных форматах. Для экспорта ноутбуков используют два формата — PDF и HTML.

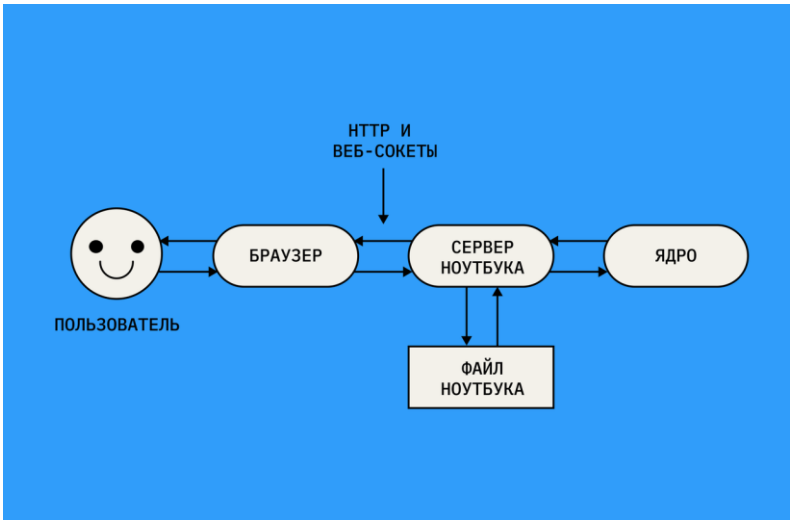
Что можно делать в веб-приложении:

● запускать и редактировать код в браузере;

● показывать результаты расчётов, используя схемы и графики;

● использовать язык разметки Markdown и LaTeX.

Один из плюсов этого инструмента в том, что код можно разделить на кусочки и работать над ними в любом порядке. Например, написать скрипт и сразу посмотреть, как он работает. Остальные фрагменты кода при этом запускать не нужно, результат появляется тут же, под частью кода. Так специалисты по Data Science выводят предварительные результаты исследований, строят графики и диаграммы.



Пользователь подключается к серверу через браузер и создаёт проект, а код отправляется через сервер в ядро. Оно запускает код и отправляет результат через сервер обратно в браузер. Сервер сохраняет проект в виде файла с расширением .ipynb
Самый простой способ набить руку в Jupyter Notebook — использовать его как калькулятор для ежедневных задач. Например, вести в блокноте семейный бюджет, сравнивать доходы и расходы.

Чтобы установить Jupyter-ноутбук и написать первый код, специальных знаний или опыта программирования на Python не нужно.

Поддерживаемые языки
Чаще всего с Jupyter-ноутбуками работают на Python, но другие языки программирования тоже поддерживаются, например R, Ruby, Pearl и даже JavaScript — всего около 40 языков.
Устанавливать их не нужно — работать с кодом из другого ядра помогают магические команды Python. Они так и называются — magic-команды и бывают двух видов:

● строчные — начинаются с %;

● ячеечные — начинаются с %%.

Чтобы, например, перейти на Ruby, нужно ввести %%ruby.

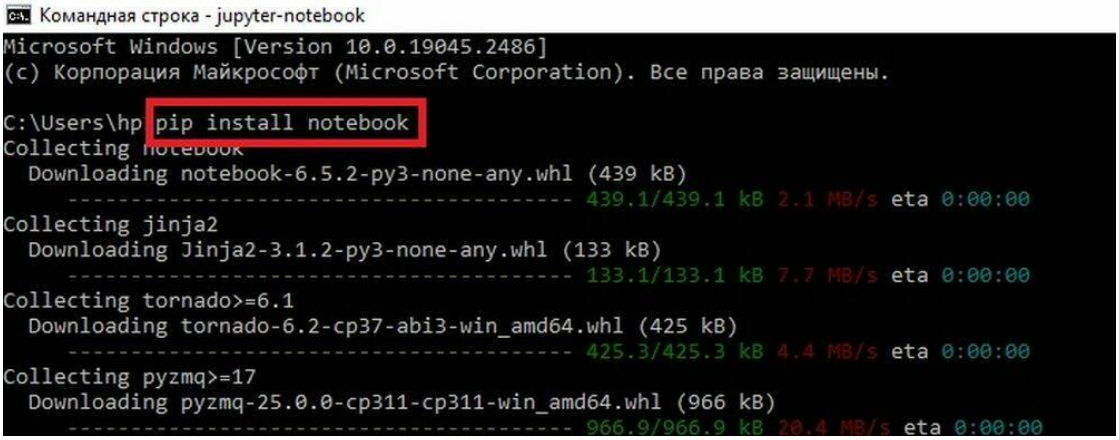
Опытные пользователи объединяют SQL и Python внутри блокнотов Jupyter. SQL хорош для извлечения данных и работы со статистикой, а Python подключается, когда нужно эти данные проанализировать. Jupyter Notebook можно подключить к любым базам данных SQL, например MySQL, Postgres, Snowflake, MariaDB, Azure, а затем писать SQL-запросы с помощью нескольких строк кода.
Jupyter-ноутбук можно запустить двумя способами: локально и в облаке.

1. Установка на локальный компьютер займёт больше времени, но код будет работать быстрее. Допустим, у пользователя Windows 10 или 11. Для начала нужно загрузить с официального сайта и установить Python 3.3 или более новую версию.

Перед установкой Python нужно поставить галочку в самом низу окна установки — без этого не получится работать с ним в командной строке

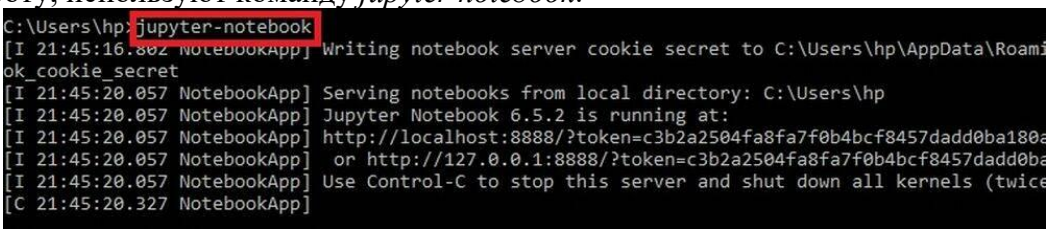
После этого нужно перейти в командную строку и установить сам ноутбук. В этом поможет специальная утилита — pip. С помощью неё можно распаковать, установить или обновить программы, в том числе и Jupyter. Утилита загружается на компьютер вместе с Python. Чтобы установить ноутбук, надо ввести команду:

pip install notebook



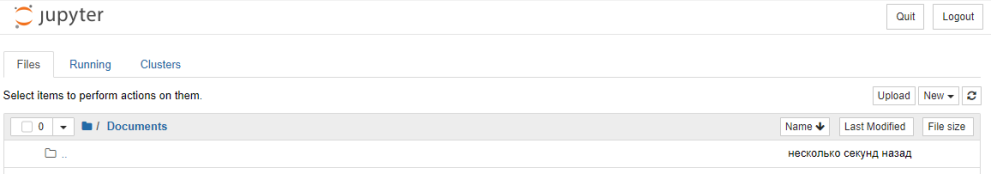
После запуска команды pip install notebook начнут загружаться файлы Jupyter — это займёт 1–2 минуты

Если выводится сообщение *Successfully installed*, то Jupyter-ноутбук готов к запуску. Чтобы начать работу, используют команду *jupyter notebook*.



Через пару секунд после ввода команды автоматически откроется браузер с интерфейсом ноутбука

Среда для разработки готова. Можно кодить.



Интерфейс ноутбука открывается прямо в браузере

Этот вариант установки подойдёт начинающим.

Для опытных специалистов есть альтернативный способ установить Jupyter-ноутбук на компьютер — загрузить Anaconda. Это дистрибутив Python и репозиторий файлов, который используют в основном для анализа данных и машинного обучения. Jupyter Notebook и его расширение JupyterLab как раз входят в этот дистрибутив.

В облаке ноутбук работает медленнее, зато можно сразу садиться и кодить. Одна из облачных версий Jupyter Notebook — программа Google Colab.

Её часто используют в учебных программах по машинному обучению. Она работает в любом браузере и не требует специальных настроек.

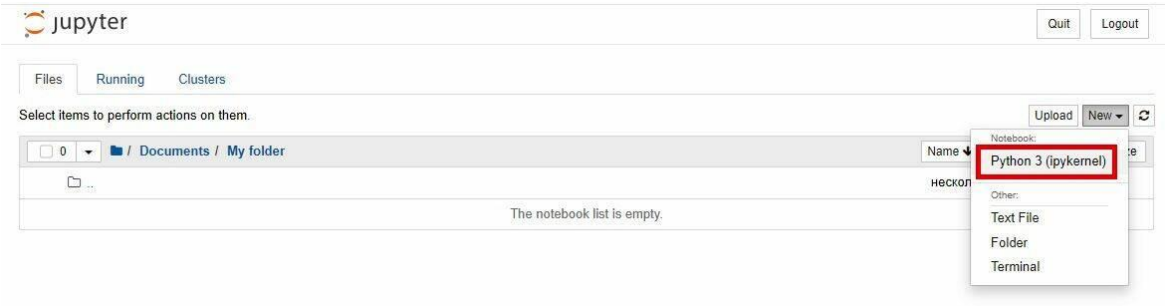
2. Работаем с ноутбуком

Разберём несколько задач, которые помогут разобраться в работе с Jupyter Notebook.

Чтобы начать новый проект, нужно запустить Jupyter-ноутбук и создать папку для проектов. Затем нажать New в правой части экрана и выбрать в списке меню Folder.

Новая папка автоматически будет названа Untitled folder. Чтобы назвать её по-другому, нужно поставить галочку напротив имени и нажать Rename

Чтобы создать ноутбук, нужно снова использовать New и выбрать Python 3.

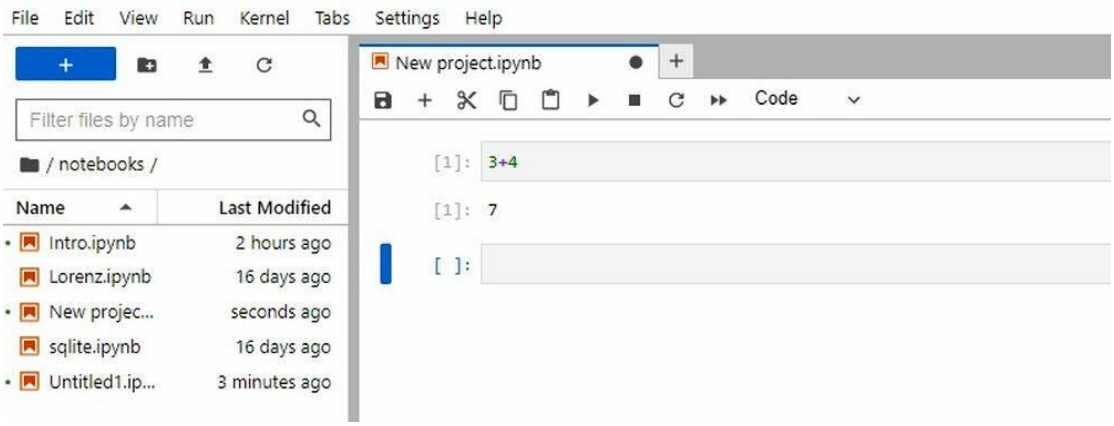


Чтобы подготовить ноутбук к работе, нужно выбрать Python 3 в выпадающем списке

Теперь можно перейти на следующий уровень и попробовать решить простую задачу — сложить 3 и 4.

Для этого нужно выставить свойство Code, ввести в ячейке 3 + 4 и нажать Ctrl + Enter или Shift + Enter. В первом случае введённый код выполнит интерпретатор Python, во

втором — будет выполнен код и появится ещё одна ячейка. Чтобы запустить код в облачной версии Jupyter, достаточно нажать Run → Run Selected Cells.



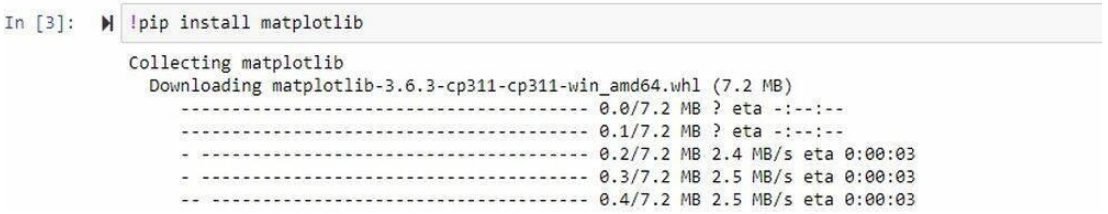
Теперь можно использовать ноутбук как калькулятор. Если получилось решить первый пример, можно потренироваться на других

Попробуем решить задачу посложнее — нарисовать график. Если ноутбук только установили, графики не получится вывести в его рабочее поле. Чтобы они отображались, нужны библиотеки pandas и Matplotlib. pandas используют для анализа данных в форме таблиц, а Matplotlib помогает строить графики и диаграммы.

Добавить библиотеки можно с помощью команд в ячейке:

```
!pip install pandas
!pip install matplotlib
```

Установка pandas займёт около минуты



Процесс установки Matplotlib ничем не отличается

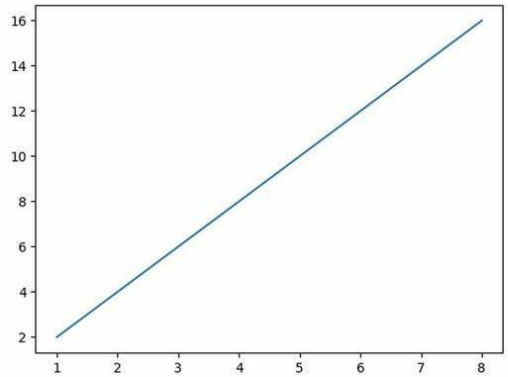
На новой странице Jupyter нужно запустить код для простого линейного графика:

```
import matplotlib.pyplot as plt
x=[1,2,3,4,5,6,7,8]
```

y=[2,4,6,8,10,12,14,16]
plt.plot(x,y)
plt.show()

Первая строка импортирует graphing-библиотеки pyplot из Matplotlib API. Третья и четвёртая строки — оси x и y. Чтобы построить график, нужно вызвать метод plot(), а для отображения — метод show().

```
import matplotlib.pyplot as plt
x=[1,2,3,4,5,6,7,8]
y=[2,4,6,8,10,12,14,16]
plt.plot(x,y)
plt.show()
```

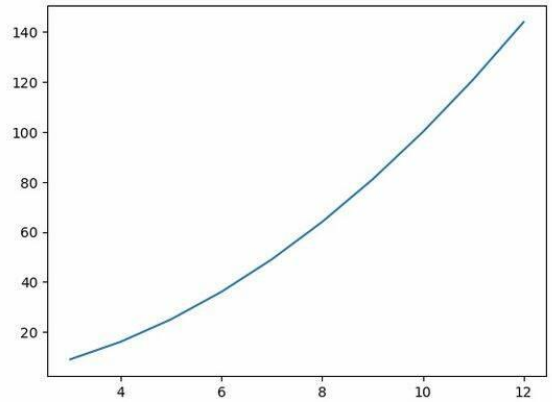


Вот так в ноутбуке выглядит линейный график

Чтобы из прямой сделать кривую, нужно подставить другие значения для оси y:

import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
plt.show()

```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
plt.show()
```



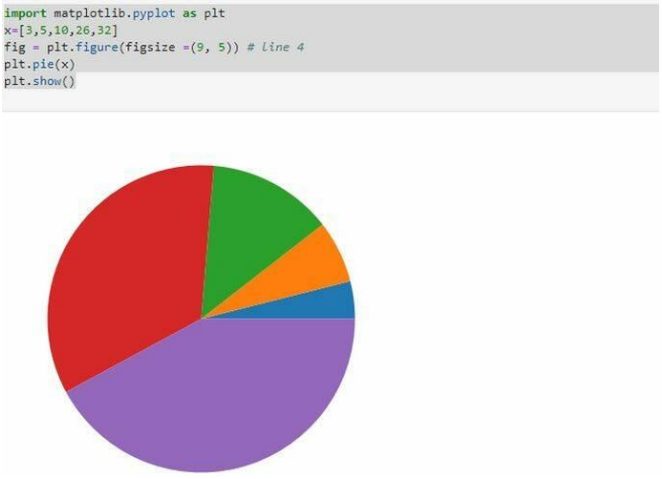
Кривую в Jupyter Notebook строить не сложнее, чем прямую, — достаточно изменить значения оси

Удобство библиотеки Matplotlib в том, что шкала рассчитывается и применяется

автоматически. Количество значений для осей x и y одинаково. Если одно из них меньше другого, при запуске кода появится ошибка, а график не отобразится.

Matplotlib подходит не только для графиков, но и для диаграмм. Здесь используется figsize — метод, который позволяет менять размеры графика. Все размеры в графиках на Jupyter-ноутбуке настраивает библиотека, поэтому визуализировать данные в нужном формате бывает сложно. Метод figsize пригодится, когда нужно, например, сделать какую-нибудь фигуру больше или меньше, изменив соотношение сторон. Официальные документы pandas советуют использовать соотношение сторон 1, но можно устанавливать любые. Например, чтобы построить диаграмму:

```
import matplotlib.pyplot as plt
x=[3,5,10,26,32]
fig = plt.figure(figsize =(9, 5)) # line 4
plt.pie(x)
plt.show()
```

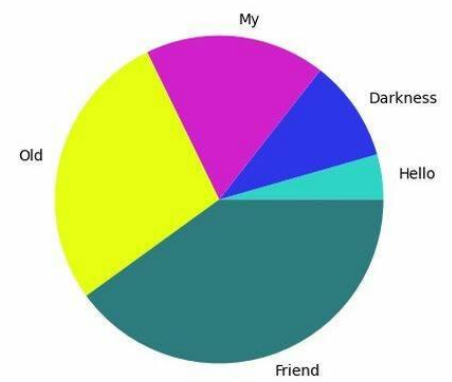


Чтобы построить круговую диаграмму, нужно использовать метод figsize

У круговой диаграммы есть два основных параметра — цвета и метки. Цвета нужны, чтобы окрашивать фрагменты диаграммы. Можно сделать это с помощью текста и прямо написать «фиолетовый» или использовать шестнадцатеричную форму (например, #7554E2). Метки используют, чтобы подписывать части диаграммы. Указание обоих параметров поможет обновить предыдущую диаграмму:

```
import matplotlib.pyplot as plt
x=[4,9,16,25,36]
fig = plt.figure(figsize =(9, 5))
plt.pie(x, labels=("Hello", "Darkness","My","Old", "Friend"),
colors = ( "#30D5C8", "#2F35E9", "#D220CD", "#E7FF12", "#2E7B80"))
plt.show()
```

```
import matplotlib.pyplot as plt
x=[4,9,16,25,36]
fig = plt.figure(figsize =(9, 5))
plt.pie(x, labels=("Hello", "Darkness","My","Old", "Friend"),
colors = ( "#30D5C8", "#2F35E9", "#D220CD", "#E7FF12", "#2E7B80"))
plt.show()
```



Изменили цвета, добавили подписи — теперь круговая диаграмма выглядит как надо

Делаем работу с Jupyter Notebook удобнее

Установить расширения для ноутбука.

По умолчанию в блокноте Jupyter нет многих функций, с которыми работать будет в разы быстрее. Но исправить это просто.

Чтобы установить расширения, нужно выполнить код в командной строке или терминале:

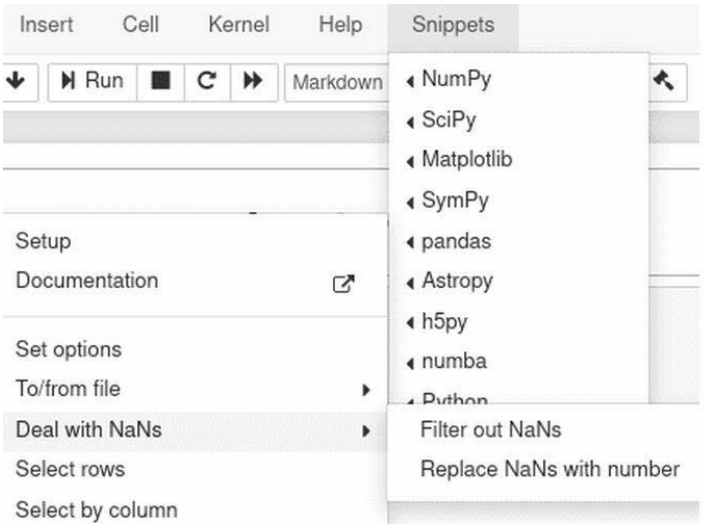
pip install jupyter_contrib_nbextensions

Затем запустить код, чтобы добавить файлы nbextensions в каталог поиска сервера Jupyter:

jupyter contrib nbextension install

В ноутбуке должна отобразиться вкладка с названием Nbextensions. Если на неё нажать, появится список расширений. Например, ярлыки для запуска нескольких ячеек, перемещение ячейки, поиск файла, автозаполнение.

Каждый раз импортировать популярные модули вроде pandas и Matplotlib или запоминать новые — это трата времени. Решить эти проблемы помогает расширение Snippets. Оно выглядит как ещё одно меню, а внутри — документация к библиотеке и доступные функции.



Цель работы: формирование навыков работы с ПО Jupyter Notebook

Рабочее задание

1. Установите на компьютер Jupyter Notebook, как было описано выше
2. Проделайте все действия, указанные в п. «Порядок выполнения работы»
3. Сохраните блокнот в файле с расширением .ipynb
4. Покажите результат преподавателю

Контрольные вопросы

1. Какие основные шаги следует предпринять для установки Jupyter Notebook на локальном компьютере?
2. Как можно изменить порт, на котором работает Jupyter Notebook?
3. Как настроить пароль для доступа к Jupyter Notebook?
4. Каковы основные элементы конфигурационного файла Jupyter Notebook?
5. Как изменить тему оформления (тему оформления) Jupyter Notebook?
6. Как добавить новое ядро (kernel) в Jupyter Notebook?
7. Как настроить Jupyter Notebook для работы с другими языками программирования, а не только с Python?
8. Как настроить Jupyter Notebook для работы с виртуальным окружением (virtualenv)?
9. Как можно управлять расширениями (extensions) в Jupyter Notebook?
10. Каким образом можно настроить Jupyter Notebook для работы с удаленным сервером?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа

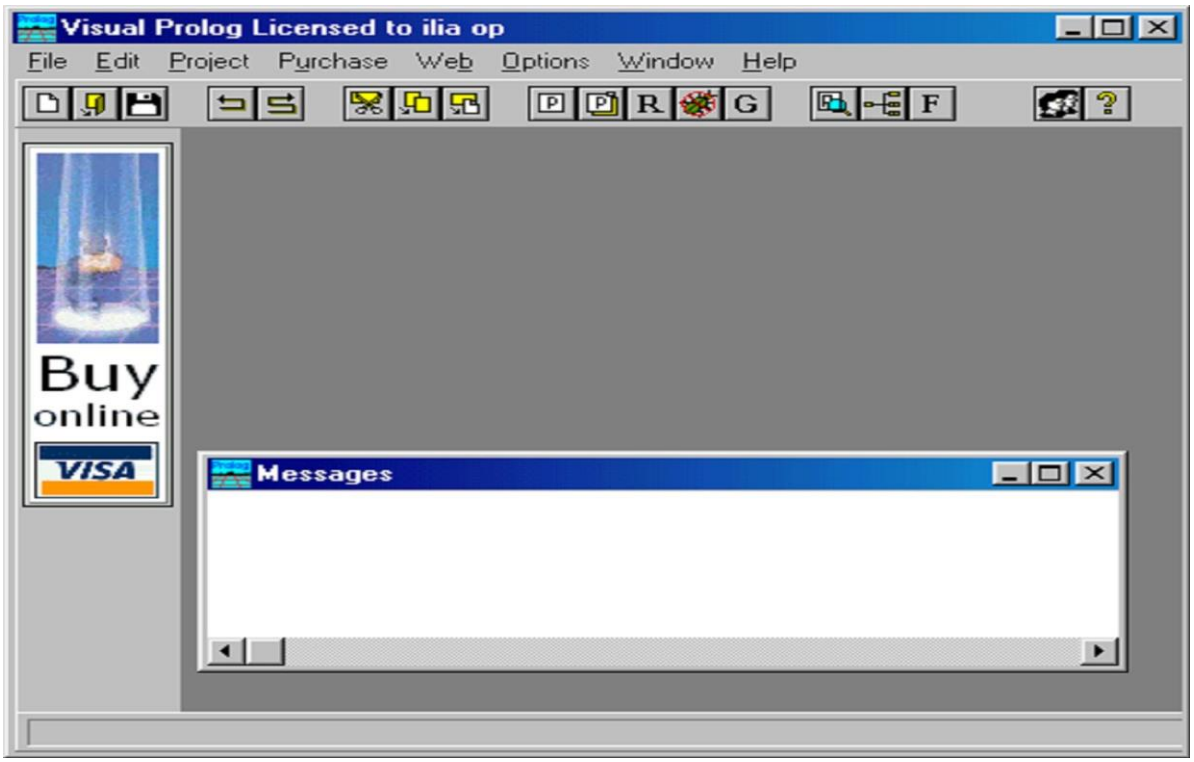
Практическое занятие №7.

Создание первой модели, определение метода создания.

Теоретическая часть

Запуск визуальной среды разработки приложений Visual Prolog.

Для того, чтобы запустить Visual Prolog, необходимо выполнить следующие действия: Пуск | Программы | Visual Prolog 5.2 | Visual Prolog 32.При этом открывается основное окно, которое называется окном **Task**



(рис. 1).

Рис. 1 Окно Task Обычно в нем доступны меню **File, Edit, Project, ptions, Help и Window**, но при fктивизации некоторых других окон в меню могут появиться дополнительные пункты.

Часто используемые команды меню могут быть выполнены и при помощи кнопок на панели инструментов (рис. 2).

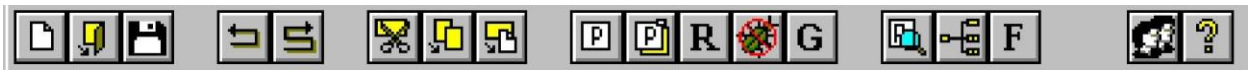


Рис. 2. Панель инструментов. Каждая из пиктограмм (табл. 1) на панели инструментов выполняет ту же функцию, что и соответствующая команда меню.Таблица 1. Команды меню и кнопки панели инструментов Кнопка панели инструментов Команда меню Кнопка панели инструментов

Команда меню File | New Edit | Copy File | Open Edit | Paste File | Save Project (Compile file) Edit | Undo Project | Build Edit | edo roject | Run Edit | Cut Project | Debug

Project | Test Goal Options Fon |t Temporary |Project | Browse Help | Local Help Project | Tree

В нижней части окна **Task,** расположена строка подсказки. Ош разделена на две части (рис.3).

Рис. 3. Строка подсказки

Левое поле используется для отображения контекстно-зависимой информа- ции, например, подсказок для командных кнопок на панели инструментов или информации о текущем элементе управления в редакторе диалоговых окон и т. д.

Крайнее правое поле используется построителем программ (make facility) для отображения состояний генерации/компиляции/компоновки текущего ресурса.

1 Создание проекта.

Для создания проекта требуется определить некоторые (не предопределенные) опции компилятора Visual Prolog. Для этого выполните следующие действия:

1. Запустите среду визуальной разработки Visual Prolog. При первом запуске VDE () проект не будет загружен, и вы увидите окно, показанное на рис. 4. Также вас проинформируют, что по умолчанию создан инициализационный файл для Visual Prolog VDE.
2. Создайте новый проект.

Выберите команду **Project | New Project,** активизируется диалоговое окно

Application Expert.

3. Определите базовый каталог и имя проекта.

Имя в поле **Project Name** следует определить как "Test". Щелкните мышью внутри поля **Name Name of .VPR File.** Также установите флажок **Mulltiprogram- mer Mode** и щелкните мышью внутри поля **Name of.PRJ File**. Вы увидите, что появится имя файла проекта **Test.prj (**рис. 4).

Рис. 4. Общие установки диалогового окна **Application Expert**

Определите цель проекта. На вкладке **Target** рекомендуется выбрать пара- метры, отмеченные на рис. 5.

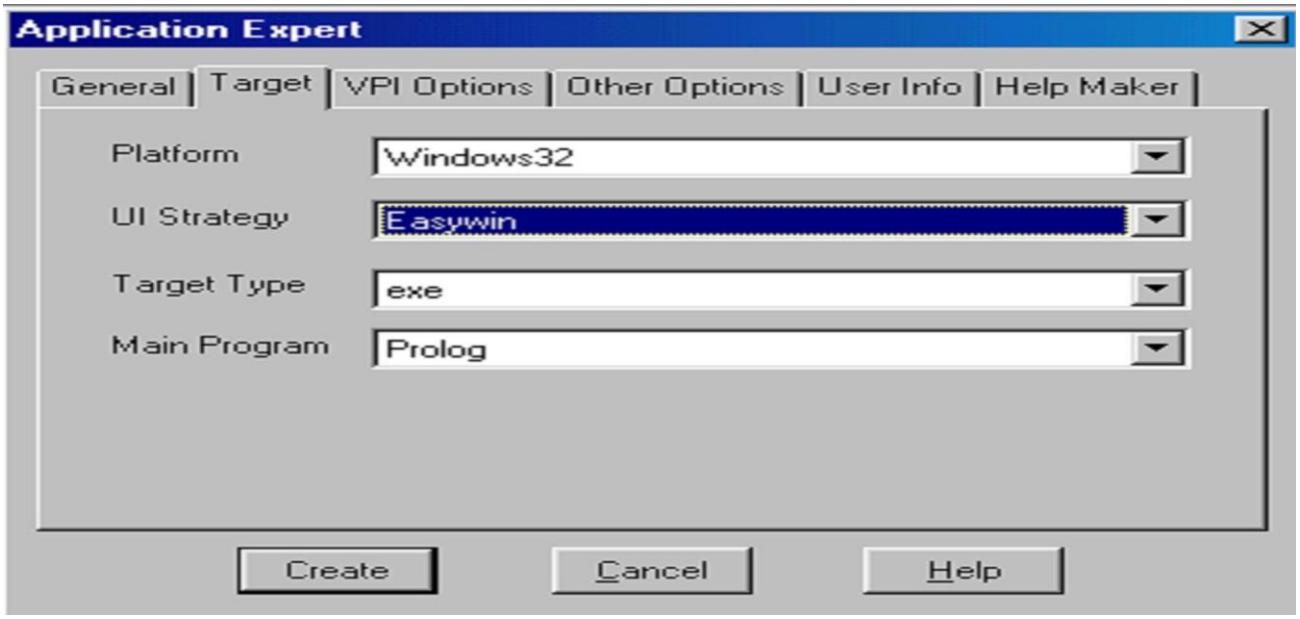


Рис. 5. Установки на вкладке Target диалогового окна **Application Expert**

Теперь нажмите кнопку **Create** для того, чтобы создать файлы проекта по умолчанию.

4. Установите требуемые опции компилятора для созданного проекта. Для активизации диалогового окна **Compiler Options** выберите команду **Options | Project | Compiler Options.** Откройте вкладку **Warnings.** Выполните следу- ющие действия:

• установите переключатель Nondeterm**.** Это нужно для того, чтобы компи- лятор Visual Prolog принимал по умолчанию, что все определенные пользо- вателем предикаты — недетерминированные (могут породить более одного решения);

• снимите флажки **Not Quoted Symbols, Strong Type Conversion Check** и **Check Type of Predicates.** Это будет подавлять некоторые возможные предупреждения компилятора;

•нажмите кнопку **ОК,** чтобы сохранить установки опций компилятора.

В результате этих действий диалоговое окно **Compiler Options** будет выгля- деть, как показано на рис. 6.

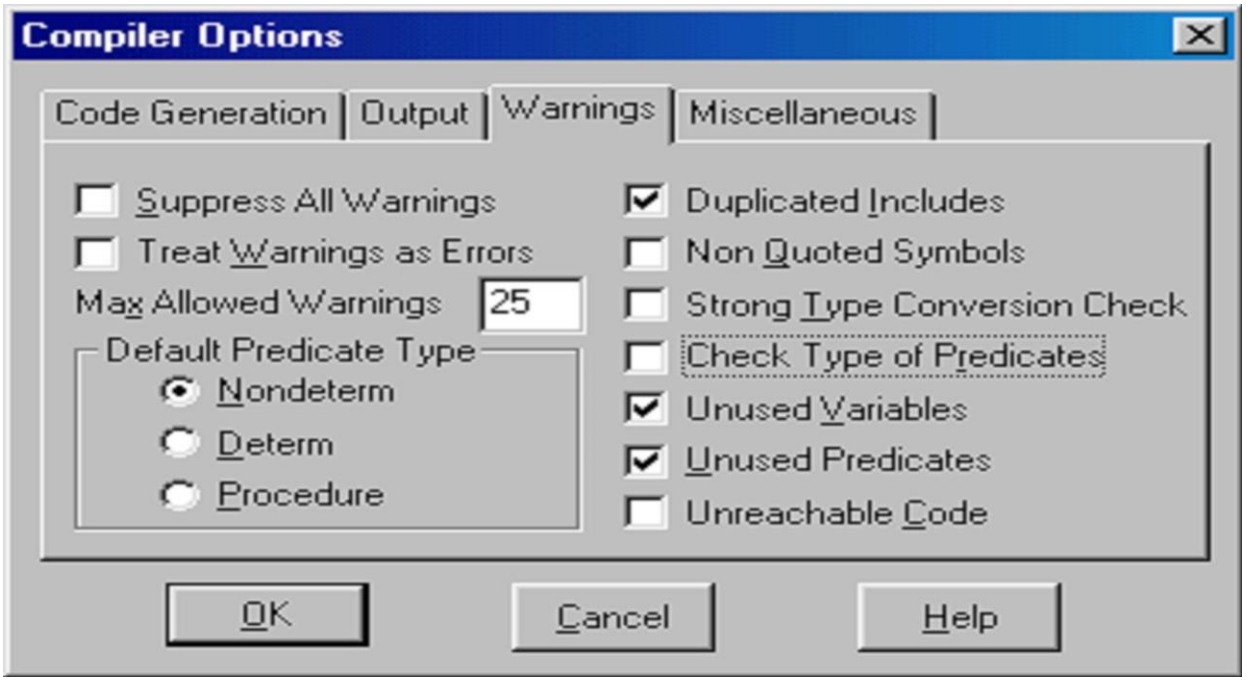


Рис. 6. Установки опций компилятора

2 Запуск и тестирование программы

Для проверки того, что ваша система настроена должным образом, следует выполнить следующие действия:

1. В окне проекта открыть файл **test.pro** (рис. 7)

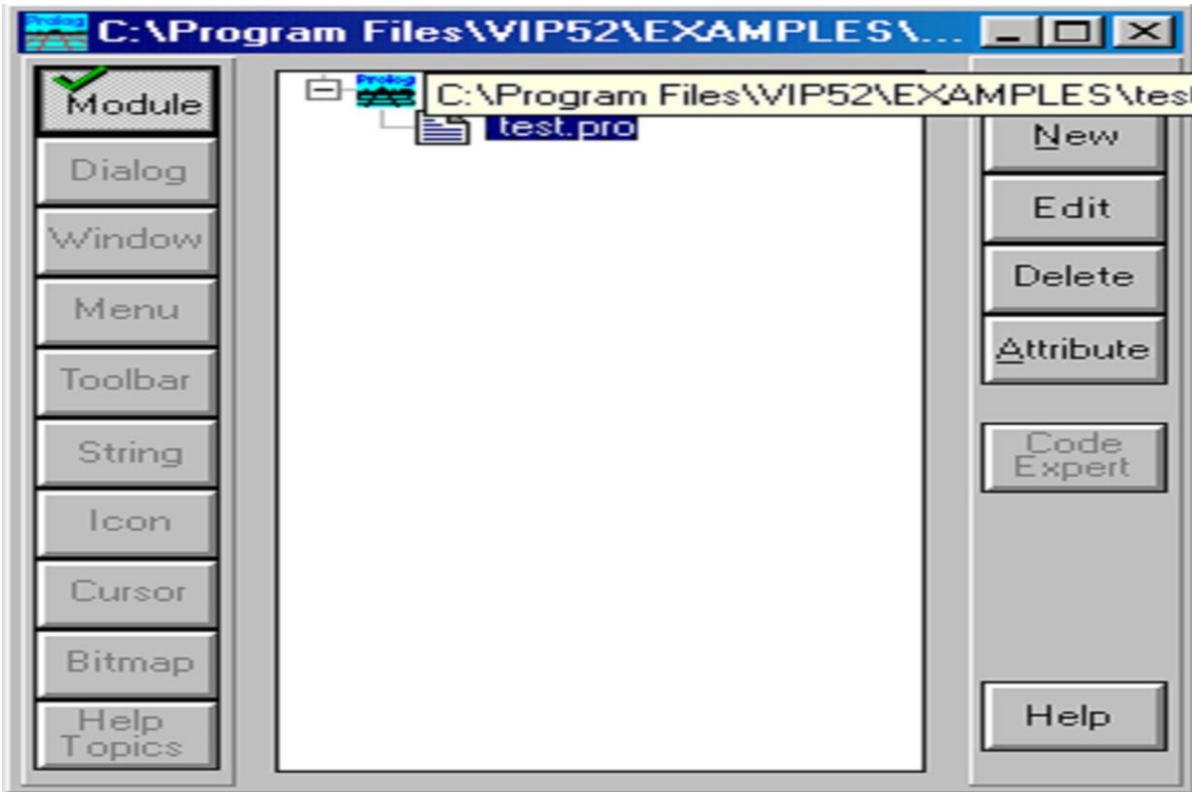


Рис. 7 Окно проекта.

2. В разделе GOAL наберите с клавиатуры write ("Hello world"), nl .

3. Нажать на панели инструментов кнопку (либо комбинацию клавиш <Ctrl>+<G>, либо активировать команду **Project | Test Goal**). В терминологии языка Пролог это называется *GOAL,* и этого достаточно для программы, чтобы она могла быть выполнена. Если ваша система установлена правильно, то экран монитора будет выглядеть, как показано на рисунке 8.

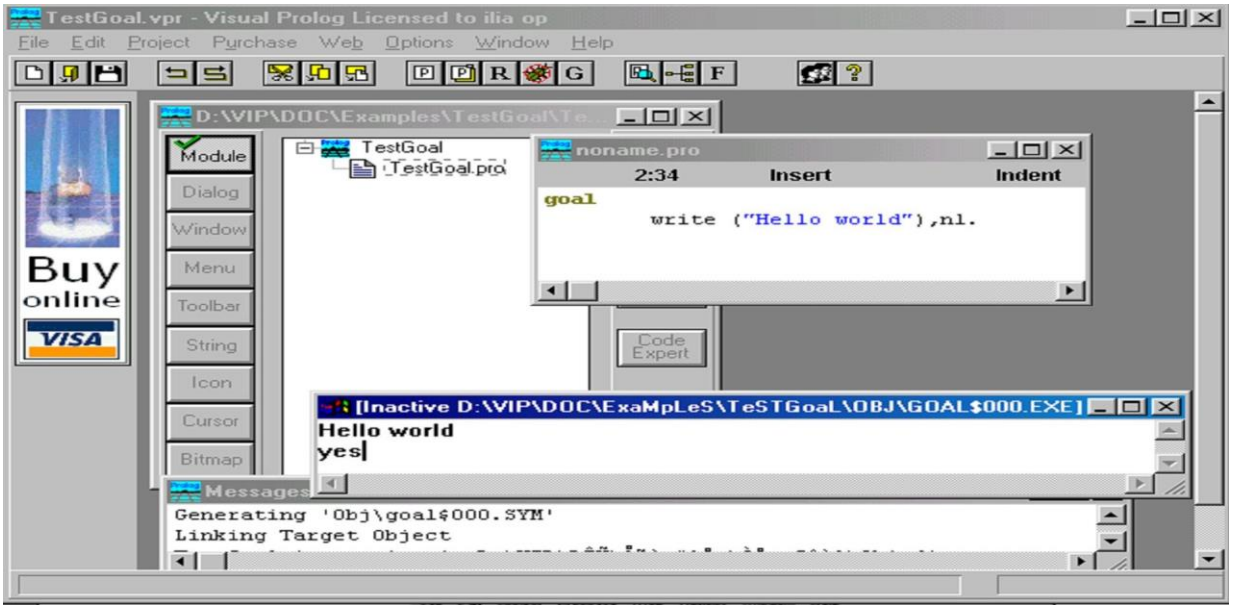


Рис. 8. Тестовая программа “Hello world”.

Результат выполнения программы будет расположен вверху в отдельном окне,

которое необходимо закрыть перед тем, как тестировать другую GOAL.

Комментарии к свойствам утилиты Test Goal

Утилита среды визуальной разработки интерпретирует GOAL как специальную программу, которая компилируется, компонуется, генерируется в исполняемый файл и Test Goal запускает его на выполнение. Эта утилита внутренне расширяет заданный код GOAL, чтобы сгенерированная программа находила все возможные решения и показывала значения всех используемых переменных. Утилита Test Goal компилирует этот код с использованием опций компилятора, заданных для открытого проекта

(рекомендуемые опции компилятора для TestGoal-проекта определили ранее).

Замечание

Утилита Test Goal компилирует только тот код, который определен в активном

окне редактора (код в других открытых редакторах или модулях проектов, если они есть, игнорируется).

При компоновке исполняемого файла TestGoal использует стратегию EASYWIN. Нельзя определить какие-либо опции компоновки для TestGoa1, т. к. игнорируются любые установки

Make Options, заданные для открытого проекта. Поэтому TestGoa1 не может использовать никакие глобальные предикаты, определенные в других модулях.

Утилита имеет ограничение на количество переменных, которые могут быть использованы в GOAL. На данный момент их 12 для 32-разрядной среды визуальной разработки, но это число может быть изменено без дополнительных уведомлений.

Обработка ошибок

Если вы допустили ошибки в программе и пытаетесь скомпилировать ее, то среда визуальной разработки отобразит окно Errors (Warnings), которое будет содержать список обнаруженных ошибок.

Дважды щелкнув на одной из этих ошибок, вы попадете на место ошибки в исходном тексте. Можно воспользоваться клавишей <F1> для вывода на экран

интерактивной справочной системы Visual Prolog. Когда окно помощи откроется, щелкните по кнопке Search, наберите номер ошибки, и на экране появится соответствующее окно помощи с более полной информацией о ней. Подробному рассмотрению основных функций интегрированной среды визуальной разработки VDE Visual Prolog посвящена следующая глава.

Команды построения

Команда Project / Compile Module

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<F9>) делает попытку компилировать модуль, содержащий редактируемый в данный момент файл. Выполнение команды зависит от следующих свойств файла:

- если файл имеет расширение pro и является модулем текущего проекта, то VDE пытается компилировать этот файл;
- если файл не является модулем текущего проекта и его расширение — pro, pre, inc, con или dom, то VDE пытается найти модуль проекта, который включает этот файл, и откомпилировать первый найденный модуль;
- во всех остальных случаях VDE пытается компилировать модуль, выбранный в окне проекта. VDE не может компилировать файл, который не является частью открытого проекта. Вместо этого файла VDE будет компилировать модуль, выбранный в окне проекта.

Если в VDE не открыт ни один проект, то никакие файлы компилироваться не будут. Команда меню Project | Compile Module заблокирована; комбинация клавиш <Ctrl>+<F9> не работает. Единственно возможное действие — это запустить утилиту Test Goal.

Команда Project / Build

Если со времени последнего построения проекта были изменены какие-либо ресурсы, то эксперты кода могут обновить некоторые секции в исходных файлах перед построением.

Эта команда (ей соответствует комбинация клавиш <Alt>+<F9>) строит проект, проверяя метки времени всех исходных файлов в проекте, поэтому если исходные файлы (или файлы, которые в них включены) являются более новыми, чем зависимые **OBJ**-файлы, то соответствующие модули проекта будут перекомпилированы.

Команда **Build** также строит файлы ресурсов и файл интерактивной справки (если необходимо). Затем проект компонуется для генерации целевого модуля (исполняемая программа или DLL).

Команда Project / Rebuild All

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<Alt>+<F9>)

выполняет то же действие, что и **Project | Build,** причем все файлы будут повторно сгенерированы или откомпилированы и скомпонованы независимо от их меток времени.

Команда Project I Stop Building

Эта команда (ей соответствует комбинация клавиш <Alt>+<F10>) используется для остановки компиляции/компоновки.

Команда Project I Run

Если необходимо, то эта команда (ей соответствует клавиша <F9>) выполнит действие **Project | Build** и затем запустит сгенерированный исполняемый файл.

Команда Project I Link Only

Эта команда (ей соответствует комбинация клавиш <Shift>+<F9>) используется для выполнения компоновки. В этом случае построитель программ вызывает компоновщика и не проверяет, нужно ли повторно компилировать какие-либо модули проекта (или даже впервые компилировать).

Команда Project / Test Goal

Эта команда (ей соответствует комбинация клавиш <Ctrl>+<G>) используется для тестирования простых целей (Goals). Программа компилируется и компонуется в специальном режиме, и затем запускается соответствующий исполняемый файл.

Утилита Test Goal ищет все решения для определенной в программе цели. Для каждого решения Test Goal отображает значения всех переменных из секции GOAL и число решений. Эта особенность — удобный способ проверить локальные предикаты в модуле.

Например, следующая цель:

GOAL X = 2; X = 1, Y = X + 1

приведет к такому результату (рис. 9):

Рис. 9 Вывод режима тестирования цели.

Команда Resource I Build Resource Only

Когда окно проекта активизировано, в меню Project появляется команда Resource. При выборе этого пункта (или нажатии комбинации клавиш

<Alt>+<F8>) генерируются выбранные файлы с расширениями rc и res и необходимые файлы констант.

Пример

Загрузите программу в среду визуальной разработки Visual Prolog и запустите ее утилитой Test Goal.

predicates
likes(symbol,symbol) clauses
likes(ellen,tennis).
likes(john,football).
likes(tom,baseball).
likes(eric,swimming).
likes(mark,tennis) .

likes(bill,Activity):-likes (tom, Activity) . goal

likes(bill, baseball).

Утилита Test Goal ответит в окне приложения: yes (да) Попробуйте также
следующий запрос в GOAL-разделе: likes(bill, tennis). Утилита Test Goal ответит:
no (нет).

Команды отладки

Команда Project | Debug

Запускает процесс отладки. Отладчик также можно запустить сочетанием клавиш CTRL+SHIFT+F9.

При помощи диалога View можно открывать дополнительные информационные окна, которые отображают различные состояния среды и переменных в режиме отладки:

View → Call Stack (Открывает информационное окно стека вызова)

View → Local Variables (Открывает информационное окно локальных переменных)

Для выполнения шагов отладки используются следующие команды: Run → Trace Intro [F7]

Run → Step Over [F8] Run → Run to Cursor [F4]

Рис. 10. Окно отладчика с открытым листингом ch02e01.pro

Задание.Подготовитьпрограмму наязыкеПРОЛОГдляполученного варианта.. Запустить программу в среде Visual Prolog в режиме отладки.

Цель работы: Освоить построение моделей в системах искусствен ного интеллекта (декларативный язык ПРОЛОГ).

Порядок выполнения работы

1. Установите на компьютер Visual Prolog, как было описано выше
2. Проделайте все действия, указанные в п. «Порядок выполнения работы»
4. Покажите результат преподавателю

Рабочее задание

1. Подготовитьпрограмму на языкеПРОЛОГдляполученного варианта.. Запустить программу в среде Visual Prolog в режиме отладки.

Контрольные вопросы

1. Какие принципы лежат в основе построения моделей в системах искусственного интеллекта с помощью декларативного языка ПРОЛОГ?
2. Какие основные концепции используются при описании знаний в ПРОЛОГе для построения моделей?
3. Какие типы фактов и правил поддерживает язык ПРОЛОГ, и как они используются при создании моделей?
4. Как можно реализовать поиск решения задачи с помощью языка ПРОЛОГ в контексте построения модели искусственного интеллекта?
5. Каким образом в ПРОЛОГе можно описывать и обрабатывать входные данные и результаты работы модели, а также осуществлять вывод информации?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 8.

Программная реализация дерева решений

Теоретическая часть

Деревья решений являются классом очень эффективной модели машинного обучения, позволяющей получить высокую точность в решении многих задач, сохраняя при этом высокий уровень интерпретации. Четкость представления информации делает деревья решений особенными среди других моделей машинного обучения. Освоенные деревом решений "знания" напрямую формируются в иерархическую структуру, которая хранит и представляет знания в понятном даже для неспециалистов виде.

Преимущества деревьев решений включают и то, что их можно использовать как для регрессии, так и для классификации, они не требуют масштабирования функций и относительно легко интерпретируются, поскольку вы можете визуализировать деревья решений. Это не только мощный способ понять вашу модель, но и рассказать, как она работает.

Мы рассмотрим классификатор из библиотеки scikit-learn на широко распространенной задаче классификации ирисов Фишера-Андерсона, описание которых дано в виде набора из данных о 150 экземплярах ириса, по 50 экземпляров из трех следующих видов (рис. 1):

- ирис щетинистый (iris setosa);
- ирис версиколор (iris versicolor);
- ирис виргинский (iris virginica).

setosa / щетинистый versicolor / версиколор virginica / виргинский

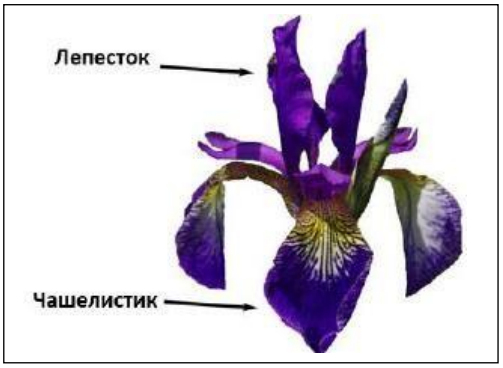
Рис. 1

Для каждого экземпляра приведены 4 следующие характеристики (в см.):

- длина чашелистика – отдельной части чашечки цветка (sepal length);
- ширина чашелистика (sepal width);
- длина лепестка (petal length).
- ширина лепестка (petal width).

Таблица – фрагмент набора данных

Длина чашелистика	Ширина чашелистика	Длина лепестка	Ширина лепестка	Вид ириса
5.1	3.5	1.4	0.2	*setosa*
4.9	3.0	1.4	0.2	*setosa*
4.7	3.2	1.3	0.2	*setosa*
…	…	…	…	…
7.0	3.2	4.7	1.4	*versicolor*
6.4	3.2	4.5	1.5	*versicolor*
6.9	3.1	4.9	1.5	*versicolor*
…	…	…	…	…
6.3	3.3	6.0	2.5	*virginica*
5.8	2.7	5.1	1.9	*virginica*
7.1	3.0	5.9	2.1	*virginica*
…	…	…	…	…



Порядок выполнения работы

Описание программы "Построение дерева решений на известномнаборе данных "Ирисы" Первое дерево создадим на классическом наборе данных Iris.

4-шаговый шаблон моделирования деревьев решений на Scikit-learn

```
Шаг 1. Импортируйте модель, которую вы хотите использовать
sklearn.tree import DecisionTreeClassifier


Шаг 2. Создайте экземпляр модели

clf = DecisionTreeClassifier (max_depth = 2, random_state = 0)


Шаг 3:
```

Нам кроме, библиотеки `sklearn` понадобятся и другие библиотеки, которые вы уже изучили.

1. Импорт библиотек

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
import pandas as pd
import numpy  as
np from sklearn
import tree
```

`train_test_split` – метод `train_test_split` библиотеки `sklearn` для разделениянабора данных на обучающую и тестовую выборки.

Набор данных `Iris` – это один из наборов данных, который поставляется с `scikit-learn`, который не требует загрузки какого-либо файла с какого-либо внешнего веб-сайта.

```
data = load_iris()
df = pd.DataFrame(data.data, columns = data.feature_names)
```

Набор данных по признакам: x_1- длина чашелистика, x_2-ширина чашелистика, x_3- длина

Набор данных по отклику: y – вид ириса

2. Загрузка набора данных по ирисам

Ядром библиотеки `pandas` являются две структуры данных, в которых происходят все операции:

- `Series`
- `Dataframes`

`Series` — это структура, используемая для работы с последовательностью одномерных данных, а `Dataframe` – более сложная и подходит для нескольких измерений (n-мерные массивы).

Ниже приведена таблица с первыми 5-ю строками набора данных, содержащего 150 строк (примеров).

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

3. Разделение набора данных на обучающие и тестовые выборки.

Приведенный ниже код помещает 75% данных в обучающую выборку и 25% данных втестовую выборку.

```
X_train, X_test, Y_train, Y_test =
train_test_split(df[data.feature_names], df['target'],
random_state=0)
```

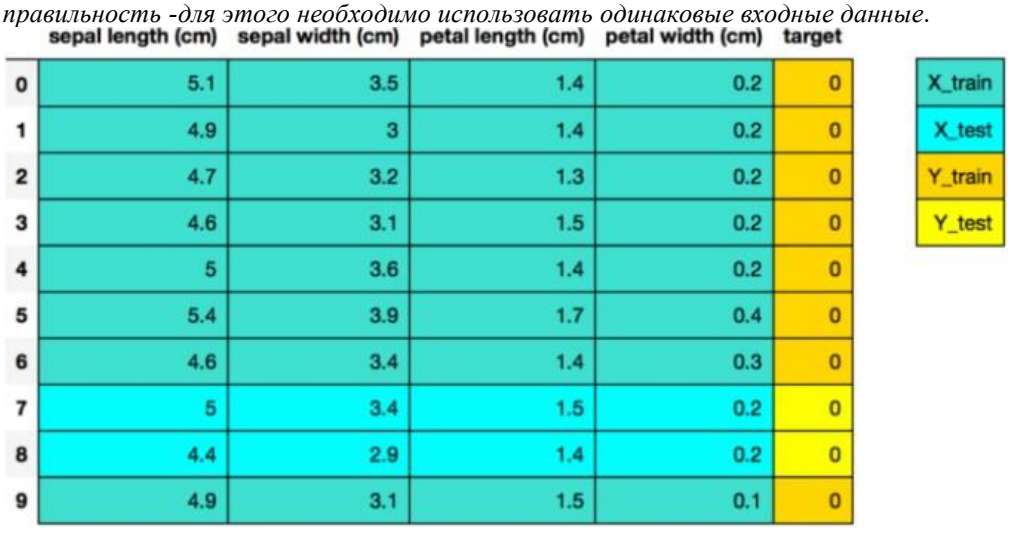
Примечание: *Суть параметра* `random_state` (*во всех функциях и методах из* `SciKit-Learn`) *в воспроизводимых случайных значениях. Т.е. если явно задать значение* `random_state` *отличным от* `None` *- то генерируемые псевдослучайные величины будут иметь одни и те же значения при каждом вызове.*

Зачем это нужно?

В задачах машинного обучения и не только часто используется генератор псевдослучайных чисел для инициализации различных параметров, весов в нейросетях, случайного разделения дата сета на обучающий проверочный сеты.

Соответственно если мы хотим сравнить несколько методов или разные наборы параметров, то длячестного сравнения надо использовать одинаковые обучающие и проверочные сеты.

Также бывает полезно создать наборы данных случайным, но воспроизводимым способом. Например, вы создали несколько различных вычислительных методов и хотите их сравнить или проверить правильность -для этого необходимо использовать одинаковые входные данные.



Цвета на изображении показывают, в какую переменную (`X_train`, `X_test`, `Y_train`, `Y_test`) были отправлены данные из фрейма данных `df` для конкретного разбиения теста поезда.

4. Создание экземпляра модели, как собственной модели классификатора

```
clf = tree.DecisionTreeClassifier(max_depth = 3, random_state = 1)
```

`max_depth = 3` – максимальная глубина дерева.

`random_state` – Управляет случайностью оценщика. Функции всегда случайным образом переставляются при каждом разделении, даже если splitter установлено значение "best". Когда алгоритм будет выбирать случайным образом в каждом разбиении, прежде чем найти наилучшее разбиение среди них. Но наилучшее найденное разделение может варьироваться в зависимости от разных прогонов. В случае, когда улучшение критерия идентично для нескольких расщеплений и необходимо выбрать одно расщепление случайным образом. Чтобы получить детерминированное поведение во время подгонки, должно быть установлено целое число. Подробнее см. в Глоссарии `max_features < n_featuresmax_featuresmax_features=n_featuresrandom_state`.

5. Обучение модели

```
clf = clf.fit(X, y)
```

Метод `fit` осуществляет обучение модели классификатора

6. Проверка способности прогнозирования классов на(тестовых) данных

```
clf.predict (X_test)
```

Проверка осуществляется методом `predict`.

7. Визуализация

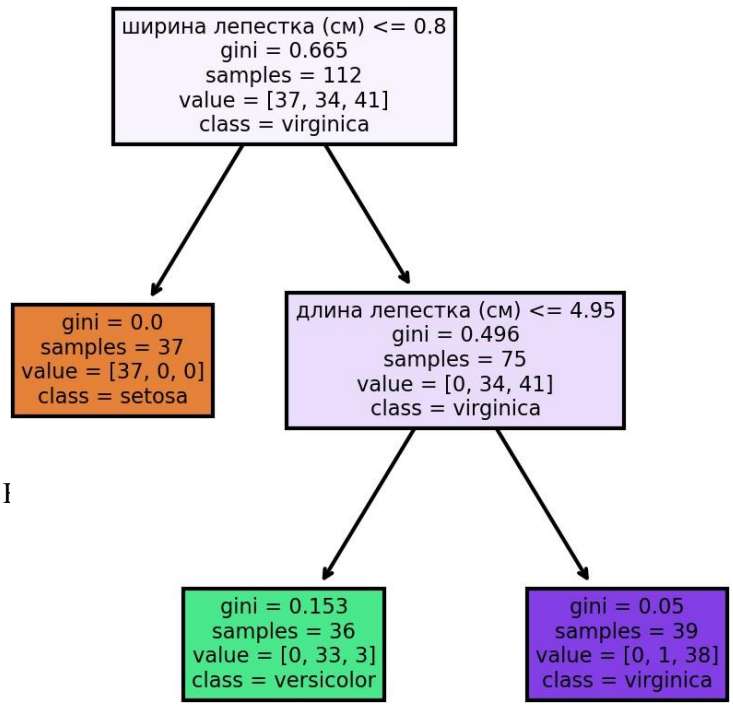
```
tree.plot_tree(clf);
```

```
fn=['sepal length (cm)','sepal width (cm)','petal length (cm)','petal
width (cm)']

cn=['setosa', 'versicolor', 'virginica']fig, axes = plt.subplots(nrows =
1,ncols = 1,figsize = (4,4), dpi=300)tree.plot_tree(clf,
            class_names=cn,
```

Полный текст программы приведен в блокноте Google Colab.

Главным результатом работы программы является дерево решений, показанное ниже:



0-й уровень, глубина 0

1-й уровень, глубина 1

2-й уровень, глубина 2

I

```
depth = 2, random_state = 0)
```

Установив `max_depth = 2`, мы задали глубину просмотра дерева, т.е. количество уровней дерева.
В этом дереве каждый узел содержит несколько характеристик:

1) условие выбора предиктора (атрибута);
2) индекс Джини;
3) общее количество примеров (`samples` = 112);
4) количество примеров по классам (`value` = [37, 34, 41]);
5) анализируемы класс (`class` = `virnica`).

Корневой узел расположен на глубине нуль, рассмотрим его:

1. В нем проверяется ширина лепестка (см) <= 0,8. Основываясь на результате, онследует либо по истинному, либо по ложному пути.
2. `Джини` = 0,665: оценка `Джини` – это показатель, который количественно определяетчистоту узла / листа. Оценка Джини больше нуля означает, что образцы,
   содержащиеся в этом узле, относятся к разным классам. Оценка Джини, равная нулю, означает, что узел чистый, что в этом узле существует только один класс выборок. В корневом узле показатель Джини больше нуля; поэтому мы знаем, что примеры, содержащиеся в корневом узле, относятся к разным классам.
3. Примеры = 112: такое количество примеров рассматривается при заданныхусловиях классификации.
4. `Значение` = [50, 50, 50]. Список говорит вам, сколько примеров в данном узле попадают в каждую категорию. Первый элемент списка показывает количество примеров, принадлежащих классу `setosa`, второй элемент списка показывает количество примеров, принадлежащих классу `versicolor`, а третий элемент списка показывает количество принадлежащих ему (`Вирджиники`) примеров. Обратите внимание, что этот узел не является чистым, поскольку в одном и том же узле содержатся разные типы классов. Мы знали это уже по индексу `Джини`.
5. `Класс` = `сетоса`: значение показывает прогноз, который сделает данный узел, и его можно определить по списку `value`. Если бы дерево решений заканчивалось в корневом узле, было бы предсказано, что все 150 выборок принадлежали к классу setosa. Конечно, это не имеет смысла, поскольку для каждого класса существует одинаковое количество выборок. Мне кажется, что дерево решенийзапрограммировано на выбор первого класса в списке, если для каждого класса имеется равное количество выборок.

Дальнейшееобъяснениеможнопосмотретьпоссылке: https://www.machinelearningmastery.ru/scikit-learn-decision-trees-explained-803f3812290d/

Цель работы получить практические навыки работы с методом деревьев решений на практических примерах с использованием языка программирования Python и библиотеки Scikit-learn

Рабочее задание

1.Изучите методический представленный методический материал пункта 2.1.
2. Откройте блокнот `google-colab` по ссылке
3.Откроется блокнот `Google Colab` "`decision tree`".
4.Откройте вкладку "`Файл`" и выберите опцию "`Сохранить копию на диске`".
Блокнот сохранится на вашем Google-диске в папке "`Colab Notebooks`", которую сам и создаст. В процессе работы каждый раз, когда вы сохраняете блокнот он будет сохраняться на вашем `Google`-диске. При доступе к блокноту по указанной ссылке вы имеете статус "читатель", после сохранения

блокнота на своем Google-диске вы становитесь его владельцем и вам автоматически присваивается статус "редактор".

5.Прогоните программу, записанную в блокноте.
6.Исследуйте программу для разной глубины дерева решений: **3**, **5**, **7**.
Отобразите в одно из деревьев и приведите полученные результаты исследований в виде таблицы:

глубина дерева	точность классификации на		класс ириса
	обучающей выборке	тестовой выборке	
2			
3			
5			
7			

Напишите в МИРО выводы по полученным результатам.

Контрольные вопросы

1. Какие признаки (features) содержатся в наборе данных "Ирисы", который часто используется для построения дерева решений?
2. Какова цель построения дерева решений на наборе данных "Ирисы" и какую задачу решает данная модель?
3. Каким образом выбирается корень дерева решений при построении на наборе данных "Ирисы"?
4. Как проводится разбиение данных на узлы дерева решений в случае набора данных "Ирисы"?
5. Каким образом происходит определение класса или категории для нового примера с использованием построенного дерева решений на данных "Ирисы"?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие №9.

Управляющие знания в системах дедукции на основе правил.

Теоретическая часть

Представление знаний с помощью правил продукции – самая распространенная форма реализации БЗ. С помощью продукции можно описать практически любую систему знаний.

Правила продукций представлены в виде импликации:

$$p_i : s_i \rightarrow d_i,$$

где p_i - правило продукции,
s_i - условие применения правила, d_i - результат применения правила.

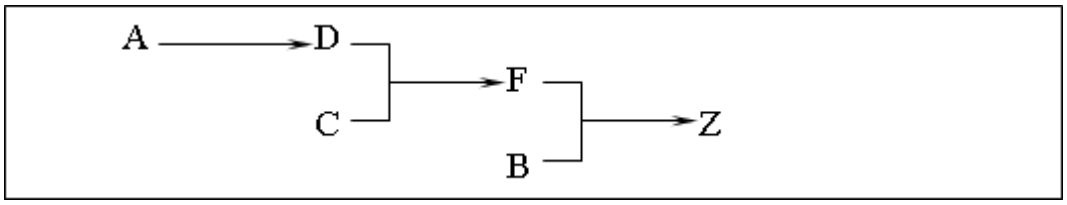


Рис.5. Пример использования правил продукции:

1. Если есть цены на выпускаемые изделия (A) - завод отпускает продукцию (D).
2. Если завод выпускает продукцию и выполняет план по ее реализации (C) - рабочие получают премию (F).
3. Если рабочие получают премию и растет производительность производства (B)- завод производит продукцию сверх плана (Z).

Рассмотрим цепочки выводов. Прямой способ рассуждения.

По известным фактам отыскивается заключение, которое следует из этих фактов и накапливается рабочая память.

Это приводит к выполнению 2 правила.

C & D$\rightarrow$ F, и факт «F» помещается в рабочую память. Тогда опять проверяются правила из базы. Первое правило выполняется $F \& B \rightarrow Z$, вследствие этого Z заносится в рабочую память. А так как Z является целью, то поиск заканчивается. Этот метод называется прямой цепочкой рассуждений, поскольку поиск новой информации происходит в направлении стрелок, разделяющих левые и правые части правил.

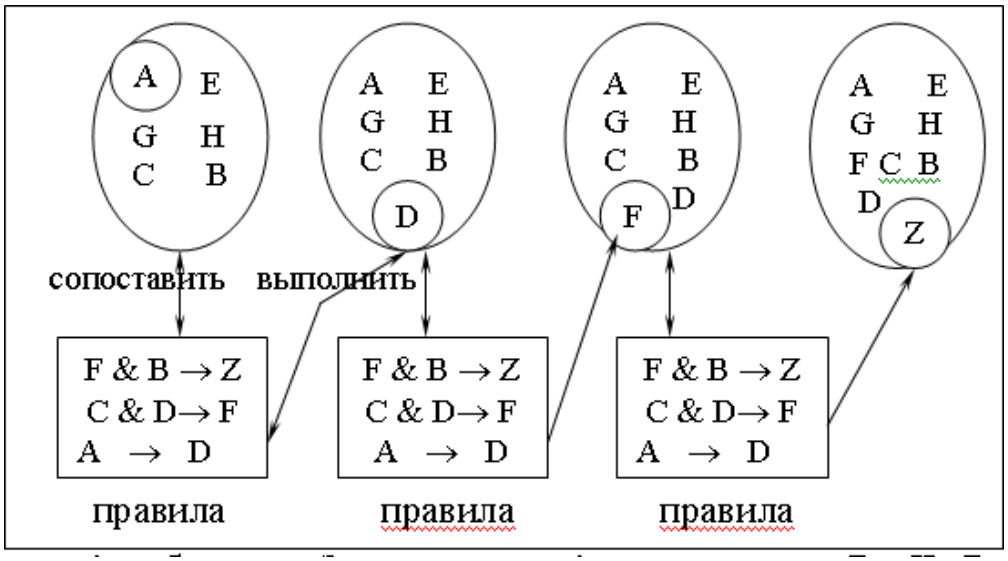


Рис.6. Пример реализации прямой цепочки рассуждений

Обобщённый алгоритм работы системы, реализующий прямую цепочку рассуждений, можно свести к следующему :

1. Определить исходное состояние.
2. Занести переменную условия в очередь переменных логического вывода, а её значение - в список переменных.
3. Просмотреть список переменных и найти ту переменную, имя которой стоит в начале очереди переменных логического вывода. Если переменная найдена, записать в указатель переменных условия номер правила и число 1. Если переменная не найдена, перейти к шагу 6.
4. Присвоить значения не проинициализированным переменным условной части найденного правила (если такие есть). Имена переменных содержатся в списке переменных условия. Проверить все условия правила и в случае их истинности обратиться к части ТО правила.
5. Присвоить значение переменной, входящей в часть ТО правила, и поместить её в конец очереди переменных логического вывода.
6. Удалить переменную, стоящую в начале очереди переменных логического вывода, если она больше не встречается в условной части какого-либо правила.

Закончить процесс рассуждений, как только опустеет очередь переменных логического вывода. Если же в очереди ещё есть переменные, вернуться к шагу 3.

Цель работы: Научиться использовать метод правил продукции для представления знаний на основе прямой цепочки рассуждений

Порядок выполнения работы

1.Изучить теоретическую часть по приведенным выше данным и дополнительной литературе.
2.Просмотреть демонстрационный пример.
3.Получить у преподавателя вариант задания для выполнения.
4.Построить прямую цепочку рассуждений
5.Реализовать программу для прямой цепочки рассуждений:

Рабочее задание

Варианты заданий

Реализовать прямую цепочку рассуждений для следующих задач:

1. прогнозирование неисправностей электронной аппаратуры
2. прогнозирование неисправностей автомобиля
3. прогнозирование заболеваний (по выбору)
4. прогнозирование (по выбору)
    a. спортивных мероприятий
    b. телепередач
    c. природных катаклизмов и т.п.
5. классификация объектов (по выбору)
6. задачи информационно-советующего характера (по выбору)
    a. помощник заведующего склада
    b. помощник аптекаря
    c. помощник оператора справочной службы
    d. выбор должности
    e. проведение отпуска и т.п.

Контрольные вопросы

1. Что такое правила продукции и в чем их сущность?
2. В чем отличие прямой цепочки рассуждений от обратной цепочки рассуждений?
3. Из каких частей состоит продукционная система?
4. Значение и применение частей продукционной системы для представления знаний?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 10.

Формальные лингвистические модели. Синтаксические анализаторы

Теоретическая часть

Лексема - минимальная единица языка, имеющая самостоятельный смысл. Например, для языка C++ лексеммами являются ‘main’, ’38.9e-4’, ‘{‘ и т.п.

Синтаксический анализатор предназначен для распознавания синтаксической структуры из представленных лексических композиций (т.е. потока лексемм в том порядке, в каком они поступают на вход синтаксического анализатора). Проще говоря (опять в качестве примера возьмём C++), синтаксический анализатор должен «проглотить» последовательность лексемм **int a = 5;** и отказаться принимать такую последовательность тех же самых лексемм: **a = 5 int;**

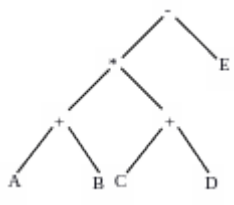
Ниже будет рассмотрен один из множества алгоритмов синтаксического анализа арифметических выражений, позволяющий вычислить их значение.

Вычисление значений выражений с помощью обратной польской нотации

Из большого числа методов и алгоритмов вычисления значений арифметических выражений наибольшее распространение получил метод трансляции с помощью обратной польской записи, которую предложил польский математик Я. Лукашевич. Следуя высказыванию Ньютона, что "в изучении наук примеры важнее правил", рассмотрим этот метод на некотором обобщённом примере.

Пусть задано простое арифметическое выражение вида:

(A+B)*(C+D)-E (1)



Представим это выражение в виде дерева, в котором узлам соответствуют операции, а ветвям - операнды. Построение начнем с корня, в качестве которого выбирается операция, выполняющаяся последней. Левой ветви соответствует левый операнд операции, а правой ветви - правый. Дерево выражения (1) показано на рис. 1.

Рис. 1

Совершим обход дерева, под которым будем понимать формирование строки символов из символов узлов и ветвей дерева. Обход будем совершать от самой левой ветви вправо и узел переписывать в выходную строку только после рассмотрения всех его ветвей. Результат обхода дерева имеет вид:

AB+CD+*E- (2)

Характерные особенности выражения (2) состоят в следовании символов операций за символами операндов и в отсутствии скобок. Такая запись называется *обратной польской записью*.

Обратная польская запись обладает рядом замечательных свойств, которые превращают ее в идеальный промежуточный язык при трансляции. Во-первых, вычисление выражения, записанного в обратной польской записи, может проводиться путем однократного просмотра, что является весьма удобным при генерации объектного кода программ. Например, вычисление выражения (2) может быть проведено следующим образом:

# п/п	Анализируемая строка	Действие
0	A B + C D + * E -	r1=A+B
1	r1 C D + * E -	r2=C+D
2	r1 r2 * E -	r1=r1*r2
3	r1 E -	r1=r1-E
4	r1	Вычисление окончено

Здесь r1, r2 - вспомогательные переменные.

Во-вторых, получение обратной польской записи из исходного выражения может осуществляться весьма просто на основе простого алгоритма, предложенного Дейкстрой. Для этого вводится понятие стекового приоритета операций (табл. 1):

Операция	Приоритет
(	1
)	1
+ | -	2
* | /	3
^	4

Таблица 1

Просматривается исходная строка символов слева направо, операнды переписываются в выходную строку, а знаки операций заносятся в стек на основе следующих соображений:

a. если стек пуст, то операция из входной строки переписывается в стек;
b. операция выталкивает из стека все операции с большим или равным приоритетом в выходную строку;
c. если очередной символ из исходной строки есть открывающая скобка, то он проталкивается в стек;
d. закрывающая круглая скобка выталкивает все операции из стека до ближайшей открывающей скобки, сами скобки в выходную строку не переписываются, а уничтожают друг друга.

Процесс получения обратной польской записи выражения (1) схематично представлен на Рис. 2:

Просматриваемый символ	1	2	3	4	5	6	7	8	9	10	11	12	13	
Входная строка	(	A	+	B	)	*	(	C	+	D	)	-	E	
Состояние стека	(	(	+ (	+ (		*	(*	(*	+ (*	+ (*	*	-	-	
Выходная строка		A		B	+			C		D	+	*	E	-

Рис. 2

Цель работы: Построение простейшего синтаксического анализатора выражений

Порядок выполнения работы

Написать программу-аналог калькулятора

Входные данные: произвольная строка символов.

Выходные данные: заключение о корректности входной строки как арифметического выражения, вычисленное значение этого выражения, в случае некорректной входной строки информацию о типе ошибки (например, «Несоответствие количества (и)»). Выходные данные зависят от сложности задания.

Синтаксический анализатор должен распознавать следующие лексеммы:

1. Знаки арифметических операций: ‘+’, ‘-‘, ‘*’, ‘/’
2. Скобки ‘(‘ и ‘)’
3. Действительные числа (т.е. дробные). Например: 12, 66.6, .54, 221.
4. Имя встроенной функции (см. задание максимум)

:

Рабочее задание

1. **Задание минимум :** считать строку символов, дать заключение о её соответствии арифметическому выражению, в случае несоответствия указать причину.
2. **Задание максимум :** вычислить значение введённого выражения с учётом приоритета операций. При этом необходимо обрабатывать исключительные ситуации (например, деление на 0). Кроме того, нужно реализовать поддержку одной встроенной функции от двух переменных (например, log(a, b)). При этом аргументы функции сами являются выражениями (в том числе содержат эту функцию). Должна быть реализована возможность сравнительно простого изменения встроенной функции по требованию преподавателя (например, заменить log(a, b) на pow(a, b)). Встроенная функция имеет наивысший приоритет. Скобки служат для изменения порядка действий (обычная математика).
3. Синтаксический анализ и вычисление значения выражения могут быть раелизованы с помощью любого алгоритма (например, с помощью метода рекурсивного спуска и т.п.)

Контрольные вопросы

1.Какие основные этапы представляет собой процесс синтаксического анализа текста?

2.Какие формальные лингвистические модели используются для представления синтаксической структуры предложений?

3.В чем заключается разница между синтаксическим анализом сверху-вниз (top-down) и синтаксическим анализом снизу-вверх (bottom-up)?

4.Как работают алгоритмы генерации синтаксических деревьев на основе контекстно-свободной

грамматики?
5.Какие задачи решают синтаксические анализаторы в современных системах обработки естественного языка?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 11.

Семантические модели. Неоднозначность и разрешение неоднозначности

Теоретическая часть

Семантическая сеть — это ориентированный граф, вершины которого — понятия, а дуги — отношения между ними. Узлы в семантической сети обычно соответствуют объектам, концепциям, событиям или понятиям. Любой фрагмент сети, например одна вершина, две вершины и соединяющие их дуги, называют подсетью. Логический вывод (поиск решения) на семантической сети заключается в том, чтобы найти или сконструировать подсеть, удовлетворяющую некоторым условиям.

Отношения, представляемые дугами, в семантической сети могут быть различными (таблица 2). Типы отношений выбираются в зависимости от вида семантической сети (таблица 3) и решаемой задачи.

Таблица 2. Основные виды отношений в семантических сетях.

Тип	**Описание**
Являться наследником (a-kind-of)	задает иерархические связи между классами
Являться экземпляром (is-a, например)	определяет значение, описывает конкретный объект, понятие
Это (are, есть)	может использоваться вместо связи a-kind-of в отношениях подразумевающих равенство или эквивалентность
Являться частью (has-part)	определяет структурные связи, описывает части или целые объекты
Функциональные	определяются обычно глаголами, отражают различные отношения (учить, владеть и т.д.)
Количественные	отображают количественные соотношения между вершинами (больше, меньше и т.д.)
Пространственные	отображают пространственные отношения между вершинами (близко, далеко и т.д.)
Временные	описывают временные связи между вершинами (скоро, долго, сейчас и т.д.)
Атрибутивные	описывают свойства объектов, понятий
Логические	описывают логические связи между вершинами (и, или, не)
По типу знания	
Экстенсиональные	описывает конкретные отношения данной ситуации
Интенсиональные	описывают имена классов объектов, а не индивидуальные имена объектов, связи отражают те отношения, которые всегда присущи объектам данного класса
По типу ограничений на дуги и вершины	
Простые	вершины сети не обладают внутренней структурой

Иерархические	вершины обладают внутренней структурой, в иерархической сети есть возможность разделять сеть на подсети и устанавливать отношения не только между вершинами, но и между подсетями (различные подсети, существующие в сети, могут быть упорядочены в виде дерева подсетей, вершины которого—подсети, а дуги — отношения видимости)
Динамические (сценарии)	сети с событиями
По количеству типов отношений	
Однородные	обладают только одним типом отношений
Неоднородные	количество типов отношений больше двух
По арности отношений	
Бинарные	все отношения в графе связывают ровно два понятия
N-арные	в сети есть отношения, связывающие более двух объектов

Пример решения задачи

Задача. Построить сетевую модель представления знаний в предметной области «Ресторан» (посещение ресторана).

Описание процесса решения. Для построения сетевой модели представления знаний необходимо выполнить следующие шаги:

1) Определить абстрактные объекты и понятия предметной области, необходимые для решения поставленной задачи. Оформить их в виде вершин.

2) Задать свойства для выделенных вершин, оформив их в виде вершин, связанных с исходными вершинами атрибутивными отношениями.

3) Задать связи между этими вершинами, используя функциональные, пространственные, количественные, логические, временные, атрибутивные отношения, а также отношения типа «являться наследником» и «являться частью».

4) Добавить конкретные объекты и понятия, описывающие решаемую задачу. Оформить их в виде вершин, связанных с уже существующими отношениями типа «являться экземпляром», «есть».

5) Проверить правильность установленных отношений (вершины и само отношение при правильном построении образуют предложение, например «Двигатель является частью автомобиля»).

Решение.

1) Ключевые понятия данной предметной области – ресторан, тот, кто посещает ресторан (клиент) и те, кто его обслуживают (повара, метрдотели, официанты, для простоты ограничимся только официантами). У обслуживающего персонала и клиентов есть общие характеристики, поэтому целесообразно выделить общее абстрактное понятие – человек. Продукцией ресторана являются блюда, которые заказывают клиенты.

Исходя из этого, вершины графа будут следующими: «Ресторан», «Человек», «Официант», «Клиент», «Заказ» и «Блюдо».

2) У этих объектов есть определенные свойства и атрибуты. Например, рестораны располагаются по определенным адресам, каждое блюдо из меню имеет свою цену. Поэтому добавим вершины «Адрес» и «Цена».

3) Определим для имеющихся вершин отношения и их типы, используя

таблицу 2.

Добавим знание о конкретных фактах решаемой задачи. Пусть имеется два ресторана: «Вкуснятина» и «Вкусная еда», в первом работает официантка Марина, а во втором официант Сергей. Пѐтр решил пойти в ресторан «Вкусная еда» и сделал заказ официанту на 2 блюда: картофель фри за 30 р., бифштекс за 130 р. Также известны адреса этих ресторанов и их специфика.

Исходя из этого, добавим соответствующие вершины в граф и соединим их функциональными отношениями и отношениями типа

«например или являться экземпляром». Полученный в результате граф изображен на рис. 3.

1) Осуществим проверку установленных связей. Например, возьмем вершину «Блюдо» и пройдем по установленным связям. Получаем следующую информацию: блюдо является частью заказа, примерами блюд могут служить картофель фри и бифштекс.

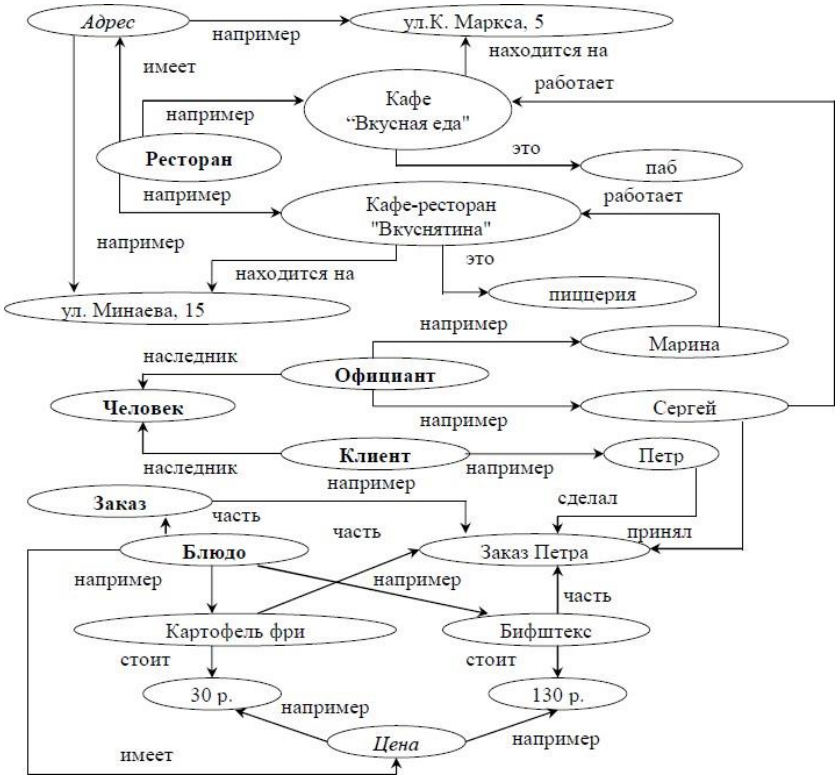


Рис. 3. Семантическая сеть предметной области «Ресторан».

Для получения ответа на какой-либо вопрос по этой задачи, необходимо найти соответствующий участок сети и, используя связи, получить результат. Например, вопрос «Какова цена заказа Петра (сколько Петр заплатил за заказ)?» Из запроса понятно, что необходимо найти следующие вершины:

«Цена», «Петр» и «Заказ» или «Заказ Петра». Часть семантической сети, находящаяся между этими вершинами, содержит ответ, а именно, частью заказа Петра являются картофель фри и бифштекс, которые стоят 30 и 130 р. соответственно. Больше информации о заказе Петра в модели нет, поэтому делаем вывод – Петр заплатил 160 р

Цель работы: Научиться использовать семантические сети для представления знаний в интеллектуальных системах.

Порядок выполнения работы

6.Изучить теоретическую часть по приведенным выше данным и дополнительной литературе;
7.Просмотреть демонстрационный пример;
8.Получить у преподавателя вариант задания для выполнения;
9.Построить семантическую модель заданного объекта;
10.Реализовать программу с использованием семантической модели
6.. Оформите отчет.

Варианты заданий

1. Используя соответствующие дуги построить семантическую сеть, касающуюся:
2. географии какого-либо региона. Дуги: государство, страна, континент, широта.
3. диагностики глазных заболеваний. Дуги: категории болезней, патофизиологическое состояние, наблюдения, симптомы.
4. распознавания химических структур. Дуги: формула вещества, свойства вещества, область применения, меры предосторожности.
5. процедуры поиска полезных ископаемых. Дуги: наименование ископаемого, расположение месторождения, глубина залегания, методы добычи.
6. судебной процедуры. Дуги: юридическое лицо, событие, меры воздействия, способы расследования.
7. распределения продуктов по магазинам. Дуги: источник снабжения, наименование продукта, способ транспортировки, конечный пункт транспортировки.
8. определения принадлежности животного к определенному виду, типу, семейству. Дуги: место обитания, строение, особенности поведения, вид питания.
9. классификации пищевых продуктов. Дуги: наименование продукта, составляющие части, способ приготовления, срок хранения.
10. распознавания типа компьютера. Дуги: страна изготовитель, стандартная конфигурация, область применения, используемое программное обеспечение.
11. иерархической структуры БД. Дуги: система, состояние, назначение, взаимодействие составляющих.

Контрольные вопросы

1.Что такое семантическая сеть и для чего ее применяют?
2.В чем состоит идея создания семантической сети?
3.Каким образом представляются данные в семантической сети?
4.Существуют ли ограничения на число связей элементов, свойств и сложность при построении семантической сети?
5.Какие отношения предложены в качестве операторов отношения для группировки вершин?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 4 часа.

Практическое занятие № 12.

Применение искусственных нейронных сетей для решения задач машинного обучения

Теоретическая часть

Краткие сведения о написании кода основных методов

В этой лабораторной работе мы углубимся в некоторые детали построения нейронной сети. Для написания кода будем использовать Python. При написании кода будем использовать две библиотеки Python – numpy для выполнения числовых вычислений и matplotlib – для визуализации результатов..Пусть имеется полносвязная нейронная сеть, имеющая структуру, изображенную на рис. 1

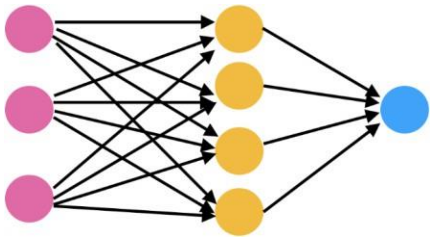


Рисунок 1 – Структура нейронной сети

Нейроны розового цвета задают вход. Нейроны желтого и голубого цветов представляют собой не что иное, как математические функции (функции активации), которые при получении неких входных данных генерируют выходные данные. Выход нейронов зависит от входа и параметров нейронов. Мы можем обновлять эти параметры, чтобы получить желаемое значение из сети.
Каждый из этих нейронов определяется с помощью сигмоидальной функции активации. сигмоидальная функция дает выходной сигнал от нуля до единицы для каждого получаемого входного сигнала. Эти сигмовидные блоки соединены друг с другом, образуя нейронную сеть.
Под соединением здесь подразумевается, что выход одного слоя сигмоидальных единиц подается в качестве входных данных для каждой сигмоидальной единицы следующего слоя. Таким образом, наша нейронная сеть выдает результат для любого заданного входа. Процесс может продолжаться до тех пор, пока мы не достигнем последнего слоя. Последний слой генерирует выходные данные.
Этот процесс нейронной сети, генерирующий выходные данные для заданных входных данных, называется прямым распространением. Выходные данные последнего слоя также называют прогнозом нейронной сети. Позже мы обсудим, как можно оценивать прогнозы. Эти оценки можно использовать, чтобы определить, нуждается ли наша нейронная сеть в улучшении или нет.
Сразу после того, как последний слой сгенерирует выходные данные, мы вычисляем функцию стоимости. Функция стоимости вычисляет, насколько далека наша нейронная сеть от желаемых

прогнозов. Значение функции стоимости показывает разницу между прогнозируемым значением и значением истинности.

Наша цель здесь — минимизировать значение функции стоимости. Процесс минимизации функции стоимости требует алгоритма, который может обновлять значения параметров в сети таким образом, чтобы функция стоимости достигла минимального значения.

Такие алгоритмы, как градиентный спуск и стохастический градиентный спуск, используются для обновления параметров нейронной сети. Эти алгоритмы обновляют значения весов и смещений каждого слоя сети в зависимости от того, как это повлияет на минимизацию функции стоимости. Влияние на минимизацию функции стоимости по отношению к каждому из весов и смещений каждого из входных нейронов в сети вычисляется методом обратного распространения ошибки.

Объяснение кода

Итак, теперь мы знаем основные идеи, лежащие в основе нейронных сетей. Давайте начнем реализовывать эти идеи в коде. Мы начнем с импорта всех необходимых библиотек.

```
import numpy as np
import matplotlib.pyplot as plt
```

Мы не будем использовать какие-либо библиотеки глубокого обучения, используем библиотеку `numpy` для эффективного выполнения математических вычислений.

Первым шагом в построении нашей нейронной сети будет инициализация параметров. Нам нужно инициализировать два параметра для каждого нейрона в каждом слое:

– вес,

– смещение.

Эти веса и смещения объявляются в векторизованной форме. Это означает, что вместо инициализации весов и смещений для каждого отдельного нейрона в каждом слое мы создадим вектор (или матрицу) для весов и еще один для смещений для каждого слоя.

Эти веса и векторы смещения будут объединены с входными данными слоя. Затем мы применим сигмоидальную функцию к этой комбинации и отправим ее в качестве входных данных на следующий слой.

Пусть переменная `Layer_dims` содержит размеры каждого слоя. Мы передадим эти размеры слоев в функцию `init_parms`, которая будет использовать их для инициализации параметров. Эти параметры будут храниться в словаре под названием `params`. Таким образом, в словаре параметров `params['W1']` будет представлять матрицу весов для слоя 1.

```
def init_params(layer_dims):
    np.random.seed(3)
    params = {}
    L = len(layer_dims)
    for l in range(1,
    L):
        params['W'+str(l)] = np.random.randn(layer_dims[l],
        layer_dims[l-1])*0.01 params['b'+str(l)] =
        np.zeros((layer_dims[l], 1))
    return params
```

Мы инициализировали веса и смещения и теперь определим сигмоидальную функцию. Он вычислит значение сигмовидной функции для любого заданного значения Z, а также сохранит это значение в виде кеша. Мы будем хранить значения кэша, потому что они нужны нам для реализации обратного распространения ошибки. Z здесь — это линейная гипотеза.

Обратите внимание, что сигмоидальная функция подпадает под класс функций активации в терминологии нейронных сетей. Задача функции активации – сформировать выходной сигнал нейрона.

Например, сигмоидальная функция принимает на вход дискретные значения и выдает значение, которое находится между нулем и единицей. Цель – преобразовать линейные выходные данные в нелинейные. Существуют различные типы функций активации, которые можно использовать для повышения производительности, но для простоты мы будем придерживаться сигмоидальной функции.

```
# Z (linear hypothesis) - Z = W*X + b,
# W - weight matrix, b- bias vector, X-
Input def sigmoid(Z):
     A = 1/(1+np.exp(np.dot(-1,
    Z))) cache = (Z)
return A, cache
```

Теперь давайте начнем писать код для прямого распространения. Ранее мы обсуждали, что прямое распространение будет брать значения из предыдущего слоя и передавать их в качестве входных данных следующему слою. Функция, приведенная ниже, будет принимать данные и параметры обучения в качестве входных данных и генерировать выходные данные для одного слоя, а затем передавать эти выходные данные на следующий уровень и так далее.

```
def forward_prop(X, params):


    A = X # input to first layer i.e. training
    data caches = []
    L = len(params)//2
    for l in range(1, L+1):
        A_prev = A
        # Linear Hypothesis
        Z = np.dot(params['W'+str(l)], A_prev) +
        params['b'+str(l)] # Storing the linear cache
        linear_cache = (A_prev, params['W'+str(l)],
        params['b'+str(l)]) # Applying sigmoid on linear
        hypothesis
        A, activation_cache = sigmoid(Z)
        # storing the both linear and activation
        cache cache = (linear_cache,
        activation_cache) caches.append(cache)
   return A, caches
```

`A_prev` — это входные данные для первого слоя. Мы пройдемся по всем слоям сети и вычислим линейную гипотезу. После этого он примет значение `Z` (линейная гипотеза) и передаст его сигмовидной функции активации. Значения `кэша` сохраняются по пути и накапливаются в `кэшах`. Наконец, функция вернет сгенерированное значение и сохраненный `кеш`.

Давайте теперь определим нашу функцию стоимости (в литературе ее принято называть функцией потерь).

```
def cost_function(A,
   Y): m = Y.shape[1]
```

```
cost = (-1/m)*(np.dot(np.log(A), Y.T) + np.dot(log(1-A), 1-
Y.T)) return cost
```

По мере уменьшения значения функции стоимости производительность нашей модели становится лучше. Значение функции стоимости можно минимизировать, обновляя значения параметров каждого из слоев нейронной сети. Такие алгоритмы, как градиентный спуск, используются для обновления этих значений таким образом, чтобы минимизировать функцию стоимости.

Градиентный спуск обновляет значения с помощью некоторых условий обновления. Эти условия обновления, называемые градиентами, рассчитываются с использованием обратного распространения ошибки. Значения градиента рассчитываются для каждого нейрона в сети и представляют собой изменение конечного результата по отношению к изменению параметров этого конкретного нейрона.

```
def one_layer_backward(dA, cache):
   linear_cache, activation_cache =
   cache

   Z = activation_cache
   dZ = dA*sigmoid(Z)*(1-sigmoid(Z)) # The derivative of the sigmoid
   function


   A_prev, W, b =
   linear_cache m =
   A_prev.shape[1]

   dW = (1/m)*np.dot(dZ, A_prev.T)
   db = (1/m)*np.sum(dZ, axis=1,
   keepdims=True) dA_prev = np.dot(W.T,
   dZ)

   return dA_prev, dW, db
```

Приведенный выше код запускает этап обратного распространения ошибки для одного слоя. Он вычисляет значения градиента для сигмовидных единиц одного слоя, используя значения кэша, которые мы сохранили ранее. В `кеше` активации мы сохранили значение `Z` для этого слоя. Используя это значение, мы рассчитаем `dZ`, который является производной функции стоимости по линейному выходу данного нейрона.

После того, как мы все это рассчитали, мы можем вычислить `dW`, `db` и `dA_prev`, которые являются производными функции стоимости по весам, смещениям и предыдущей активации соответственно. Мы напрямую использовали формулы в коде. Если вы не знакомы с исчислением, на первый взгляд оно может показаться слишком сложным. Но сейчас подумайте об этом, как о любой другой математической формуле.

После этого мы будем использовать этот код для реализации метода обратного распространения ошибки для всей нейронной сети. Функция `backprop` реализует код этого метода. Здесь мы создали словарь для сопоставления градиентов каждому слою. Мы пройдемся по модели в обратном направлении и вычислим градиент.

```
def backprop(AL, Y,
   caches): grads = {}
```

```
    L = len(caches)
    m = AL.shape[1]
    Y = Y.reshape(AL.shape)


    dAL = -(np.divide(Y, AL) - np.divide(1-Y, 1-AL))


    current_cache = caches[L-1]
    grads['dA'+str(L-1)], grads['dW'+str(L-1)],
grads['db'+str(L-1)] = one_layer_backward(dAL, current_cache)

    for l in reversed(range(L-1)):


        current_cache = caches[l]
        dA_prev_temp, dW_temp, db_temp =
one_layer_backward(grads["dA" + str(l+1)], current_cache)
        grads["dA" + str(l)] =
        dA_prev_temp grads["dW" + str(l +
        1)] = dW_temp grads["db" + str(l
        + 1)] = db_temp

    return grads
```

После того, как мы прошлись по всем слоям и вычислили градиенты, мы сохраним эти значения в словаре градиентов и вернем их.

Наконец, используя эти значения градиента, мы обновим параметры для каждого слоя. Функция `update_parameters` проходит через все слои, обновляет параметры и возвращает их.

```
def update_parameters(parameters, grads, learning_rate):
    L = len(parameters) // 2


    for l in range(L):
        parameters['W'+str(l+1)] = parameters['W'+str(l+1)] -
learning_rate*grads['W'+str(l+1)]
        parameters['b'+str(l+1)] = parameters['b'+str(l+1)] -
learning_rate*grads['b'+str(l+1)]

    return parameters
```

Наконец пришло время собрать все это воедино. Мы создадим функцию под названием `train` для обучения нашей нейронной сети.

```
def train(X, Y, layer_dims, epochs, lr):
    params = init_params(layer_dims)
    cost_history = []

    for i in range(epochs):
        Y_hat, caches = forward_prop(X,
        params) cost = cost_function(Y_hat, Y)
        cost_history.append(cost)
        grads = backprop(Y_hat, Y, caches)
```

```
        params = update_parameters(params, grads, lr)

    return params, cost_history
```

Эта функция будет выполнять все функции шаг за шагом для заданного количества эпох. После завершения он вернет окончательные обновленные параметры и историю затрат. Историю затрат можно использовать для оценки производительности вашей сетевой архитектуры.

Цель работы: сформировать у обучающихся представление о коде базовых методов библиотек Python, используемых для обучения нейронной сети

Порядок выполнения работы

–изучите первый и второй разделы настоящей лабораторной работы;
–откройте новый блокнот в Google Colab (https://colab.research.google.com/), почитайте руководство по работе с ним;
–вставьте код в блокнот GoogleColabи проверьте его;
–полученныепромежуточныерезультатыисвоивыводызапишитев интерактивной онлайн-доске Miro;
–напишите программу, реализующую решение задачи "Прогнозирование выбора друга", описанной в лекции "Полносвязные нейронные сети";
–полученные результаты и свои выводы запишите в текстовый файл

Контрольные вопросы

1.Какой принцип работы лежит в основе полносвязной нейронной сети, и какие компоненты она включает?
2.Какие шаги необходимо выполнить для создания первой модели полносвязной нейронной сети, начиная с выбора количества слоев и нейронов?
3.Какой метод оптимизации обычно используется при обучении полносвязной нейронной сети, и как настраиваются гиперпараметры для достижения оптимальной производительности?
4.Какие типичные проблемы могут возникнуть при разработке и обучении первой модели полносвязной нейронной сети, и как их можно решить или избежать?
5.Как провести оценку и тестирование первой модели полносвязной нейронной сети, а также как оценить ее эффективность и качество работы на новых данных?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 13.

Применение искусственных нейронных сетей для решения задач.

Теоретическая часть

Организация среды разработки

Систему программирования на языке Python 3.6.4 для Windows можно загрузить с официального сайта https://www.python.org/downloads/windows/ (рекомендуется использовать Windows x86 executable installer). ***Перед установкой необходимо выбрать пункт "Add Python to PATH"***.

В качестве самоучителя по языку Python можно использовать ресурс https://pythonworld.ru/samouchitel-python.

Установка библиотеки NumPy

1. Запустите приложение «Командная строка» (для этого наберите `cmd` в окне поиска панели задач Windows).

```
pip3 install numpy
```

2. Запустите команду для установки пакета:

Установка рабочей папки проекта

Создайте каталог `NeuralNetwork`, в котором Вы будете хранить исходные тексты программ, создаваемых в ходе выполнения практических заданий (например, `C:\NeuralNetwork`). В каталоге `NeuralNetwork` создайте подкаталог `Network1`, в котором будут храниться исходные коды задания 1.

Создание нейронной сети

Запустите среду разработки (для запуска среды разработки IDLE, наберите idle в окне поиска панели задач Windows). Создайте новый файл для программы (меню `File/New File`). Сохраните этот файл в каталоге `Network1` под именем `network` (меню `File/Save`). Расширение `.py` будет подставлено по умолчанию.

Скопируйте в окно программы `network.py` следующие команды и впишите свои данные:

```
"""
network.py
Модуль создания и обучения нейронной сети для распознавания рукописных цифр с использованием
метода градиентного спуска.
Группа:<Указать номер группы> ФИО:<Указать ФИО
студента> """
#### Библиотеки
# Стандартные библиотеки
```

```
import random # библиотека функций для генерации случайных значений

# Сторонние библиотеки
import numpy as np # библиотека функций для работы с матрицами

""" ---Раздел описаний--- """
""" --Описание класса Network--"""
class Network(object): # используется для описания нейронной сети def __init__(self, sizes): #
    конструктор класса
                                                # self – указатель на объект класса
                                                # sizes – список размеров слоев нейронной
сети сети

self.num_layers = len(sizes) # задаем количество слоев нейронной
self.sizes = sizes # задаем список размеров слоев нейронной сети self.biases = [np.random.randn(y, 1) for y in
sizes[1:]] # задаем

случайные начальные смещения
            self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes[1:])] # задаем случайные
начальные веса связей
""" --Конец описания класса Network--""" """ --- Конец
раздела описаний--- """
""" ---Тело программы--- """
net = Network([2, 3, 1]) # создаем нейронную сеть из трех слоев """ ---Конец тела
программы--- """
""" Вывод результата на экран: """ print('Сеть net:')
print('Количетво слоев:', net.num_layers) for i in
range(net.num_layers):
    print('Количество нейронов в слое', i,':',net.sizes[i]) for i in range(net.num_layers-
1):
    print('W_',i+1,':') print(np.round(net.weights[i],2))
    print('b_',i+1,':') print(np.round(net.biases[i],2))
```

Сохраните файл `network.py` и выполните программу `network`. Для того чтобы запустить исполнение программы, выберите `Run/Run Module` (или нажмите `F5`). В результате будет создан объект класса `Network`, задающий трехуровневую нейронную сеть с соответствующими параметрами. При создании объекта класса `Network` веса и

смещения инициализируются случайным образом. Для инициализации этих величин используется функция `np.random.randn` из библиотеки `NumPy`. Данная функция, генерирует числа с нормальным распределением для массива заданной размерности.

Определение сигмоидальной функции

В качестве функции активации для нейронов сети используется сигмоидальная функция, вычисляющая выходной сигнал искусственного нейрона. Ниже представлен код, для определения функции. Добавьте этот код в раздел описаний программы `network.py`.

```
def sigmoid(z): # определение сигмоидальной функции активации return 1.0/(1.0+np.exp(-z))
```

Обратите внимание, что для описания сигмоидальной функции активации используется функция для вычисления экспоненты из библиотеки `NumPy`, это позволяет передавать массив в качестве

входного параметра сигмоидальной функции. В этом случае функция экспоненты применяется поэлементно, то есть в векторизованной форме.

Метод feedforward

Добавьте метод `feedforward` в описание класса `Network`.

```
def feedforward(self, a):
    for b, w in zip(self.biases, self.weights): a = sigmoid(np.dot(w,
        a)+b)
```

Данный метод осуществляет подсчет выходных сигналов нейронной сети при заданных входных сигналах. Параметр a является массивом $n \times 1$, где n – количество нейронов входного слоя. Функция `np.dot` вычисляет произведение матриц. Для подсчета выходных значений нейронной сети, необходимо один раз вызвать метод `feedforward`, в результате чего выходные сигналы будут последовательно вычислены для всех слоев нейронной сети.

Обучение нейронной сети

Для реализации механизма обучения создаваемой нейронной сети добавим метод SGD, который реализует стохастический градиентный спуск. Метод имеет следующие параметры:

«`Training_data`» – обучающая выборка, состоящая из пар вида $(x\rightarrow, y\rightarrow)$, где $x\rightarrow$ – вектор входных сигналов, а $y\rightarrow$ – ожидаемый вектор выходных сигналов;

«epochs»– количество эпох обучения;

«mini_batch_size»- размер подвыборки;

«eta»- скорость обучения;

«`test_data`» - (необязательный параметр); если данный аргумент не пуст, то программа после каждой эпохи обучения осуществляет оценку работы сети и показывает достигнутый прогресс.

Добавьте программный код метода `SGD` в раздел в описания класса `Network`:

```
    def SGD( # Стохастический градиентный спуск
            self                    # указатель на объект класса
          , training_data           # обучающая выборка
          , epochs                  # количество эпох обучения
          , mini_batch_size  # размер подвыборки
          , eta                     # скорость обучения
          , test_data               # тестирующая выборка
          ):
          test_data = list(test_data) # создаем список объектов тестирующей
выборки
          n_test = len(test_data) # вычисляем длину тестирующей выборки training_data = list(training_data)
          # создаем список объектов
обучающей выборки
          n = len(training_data) # вычисляем размер обучающей выборки for i in range(epochs): #
```

Данный программный код работает следующим образом. В начале каждой эпохи обучения элементы обучающей выборки перемешиваются (переставляются в случайном порядке) с помощью функции `shuffle()` из библиотеки `random`, после чего обучающая выборка последовательно разбивается на подвыборки длины `mini_batch_size`. Для каждой подвыборки выполняется один шаг градиентного спуска с помощью метода `update_mini_batch` (см. ниже). После того, как выполнен последний шаг градиентного спуска, т.е. выполнен метод `update_mini_batch` для последней подвыборки, на экран выводиться достигнутый прогресс в обучении нейронной сети, вычисляемый на тестовой выборке с помощью метода `evaluate` (см. ниже).

Анализируя программный код метода `update_mini_batch` можно увидеть, что основная часть вычислений осуществляется при вызове метода `backprop` (см. ниже). Данный метод класса `Network` реализует алгоритм обратного распространения ошибки, который является быстрым способом вычисления градиента стоимостной функции. Таким образом, метод `update_mini_batch` вычисляет градиенты для каждого прецедента (x, y) в подвыборке, а затем соответствующим образом обновляет веса и смещения

нейронной сети. Добавьте код метода `update_mini_batch` в раздел в описания класса `Network`:

```
    def update_mini_batch( # Шаг градиентного спуска self# указатель на
            объект класса

            , mini_batch                # подвыборка

            , eta                       # скорость обучения

            ):

        nabla_b = [np.zeros(b.shape) for b in self.biases] # список градиентов dC/db для каждого слоя
(первоначально заполняются нулями)

        nabla_w = [np.zeros(w.shape) for w in self.weights] #список градиентов dC/dw для каждого слоя
(первоначально заполняются нулями)

        for x, y in mini_batch:

            delta_nabla_b, delta_nabla_w = self.backprop(x, y) # послойно вычисляем градиенты dC/db и
dC/dw для текущего прецедента (x, y)

            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)] # суммируем градиенты dC/db
для различных прецедентов текущей подвыборки
```

Скопируйте в раздел описания класса `Network` программный код метода `backprop`, реализующего алгоритм обратного распространения:

```
    def backprop( # Алгоритм обратного распространения self# указатель
            на объект класса
            , x         # вектор входных сигналов
            , y         # ожидаемый вектор выходных сигналов
            ):
        nabla_b = [np.zeros(b.shape) for b in self.biases] # список градиентов dC/db для каждого слоя
(первоначально заполняются нулями)
        nabla_w = [np.zeros(w.shape) for w in self.weights]# список градиентов dC/dw для каждого
слоя (первоначально заполняются нулями)
        # определение переменных
        activation = x # выходные сигналы слоя (первоначально соответствует выходным сигналам 1-го
слоя или входным сигналам сети)
        activations = [x] # список выходных сигналов по всем слоям (первоначально содержит только
выходные сигналы 1-го слоя)
        zs = [] # список активационных потенциалов по всем слоям (первоначально пуст)
        # прямое распространение
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b # считаем активационные потенциалы текущего слоя
            zs.append(z) # добавляем элемент (активационные потенциалы слоя) в конец списка
            activation = sigmoid(z) # считаем выходные сигналы текущего слоя, применяя
сигмоидальную функцию активации к активационным потенциалам слоя
            activations.append(activation) # добавляем элемент (выходные сигналы слоя) в конец списка
```

```
        # обратное распространение
        delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1]) # считаем меру влияния
нейронов выходного слоя L на величину ошибки (BP1)
        nabla_b[-1] = delta # градиент dC/db для слоя L (BP3)
        nabla_w[-1] = np.dot(delta, activations[-2].transpose()) # градиент dC/dw для слоя L (BP4)

        for l in range(2, self.num_layers):
            z = zs[-l] # активационные потенциалы l-го слоя (двигаемся по списку справа налево)
            sp = sigmoid_prime(z) # считаем сигмоидальную функцию от активационных потенциалов
l-го слоя
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp # считаем меру влияния нейронов l-
го слоя  на величину ошибки (BP2)
            nabla_b[-l] = delta # градиент dC/db для l-го слоя (BP3) nabla_w[-l] = np.dot(delta,
            activations[-l-1].transpose())#

градиент dC/dw для l-го слоя (BP4) return (nabla_b,
        nabla_w)
```

Скопируйте в раздел описания класса `Network` программный код метода `evaluate`, демонстрирующего прогресс в обучении:

```
    def evaluate(self, test_data): # Оценка прогресса в обучении test_results =
        [(np.argmax(self.feedforward(x)), y)

                                    for (x, y) in test_data]
```

Указанный метод возвращает количество прецедентов тестирующей выборки, для которых нейронная сеть выдает правильный результат. Тестирующая выборка состоит из пар (x, y), где x – вектор размерности 784, содержащий изображение цифры, а y – целое числовое значение цифры, изображенной на картинке. Ответ нейронной сети определяется как номер нейрона в выходном слое, имеющего наибольшее значение функции активации. Метод `evaluate` вызывается в методе `SGD` после завершения очередной эпохи обучения.

Скопируйте в раздел описания класса Network программный код метода `cost_derivative`, вычисляющего вектор частных производных $\mathrm{AC}(a^L) = a^L - y$:

```
    def cost_derivative(self, output_activations, y): # Вычисление частных производных стоимостной
функции по выходным сигналам последнего слоя

        return (output_activations-y)
```

Указанный метод вызывается в методе `backprop`.

Скопируйте в конец раздела описаний (после функции `sigmoid`) код функции `sigmoid_prime`, вычисляющей производную сигмоидальной функции:

```
def sigmoid_prime(z):# Производная сигмоидальной функции

    return sigmoid(z)*(1-sigmoid(z))
```

□ Сохраните и закройте файл `network.py`.

Работа с базой данных MNIST

Для обучения нейронной сети будем использовать архив http://deeplearning.net/data/mnist/mnist.pkl.gz с сайта Лаборатории машинного обучения Университета Монреаля, сформированный на основе базы данных MNIST, который содержит 70 000 изображений рукописных цифр, разделенных на три набора:

1) `training_data` – набор из 50 000 изображений предназначен для обучения нейронных сетей;
2) `validation_data` – набор из 10 000 изображений предназначен для текущей оценки работы алгоритма обучения и подбора параметров обучения (используется в последующих лабораторных работах);
3) `test_data` – набор из 10 000 изображений предназначен для проверки работы нейронной сетей.

Каждый набор состоит из двух списков: списка изображений (в градациях серого) и соответствующего списка цифр в диапазоне от 0 до 9. Изображение представлено в виде одномерного numpy-массива размера $784 = 28 \times 28$ значений от 0 до 1, где 0 соответствует черному цвету пиксела, а 1 – белому.

- ☐ **Загрузите архив http://deeplearning.net/data/mnist/mnist.pkl.gz и сохраните его в директории Network1.**

Функции для работы с базой данных MNIST целесообразнее вынести в отдельный файл. Создайте новый файл `mnist_loader` и сохраните его в директории `Network1`. Скопируйте в окно программы `mnist_loader.py` следующие команды и впишите свои данные:

```
"""

mnist_loader.py

~~~~~~~~~~

Модуль для подключения и использования базы данных MNIST.

Группа:[Указать номер группы]
ФИО:[Указать ФИО студента] """

import gzip # библиотека для сжатия и распаковки файлов gzip и gunzip. import pickle # библиотека для сохранения и загрузки сложных объектов Python.

import numpy as np # библиотека для работы с матрицами def load_data():

    f = gzip.open('mnist.pkl.gz', 'rb') # открываем сжатый файл gzip в двоичном режиме

    training_data, validation_data, test_data = pickle.load(f, encoding='latin1') # загружам таблицы из файла

    f.close() # закрываем файл
```

Для использования базы данных MNIST в нашей программе необходимо скорректировать форматы наборы training_data, validation_data и test_data. Это делается в функции `load_data_wrapper`. Скопируйте в файл `mnist_loader` следующий программный код.

```
def load_data_wrapper():



    tr_d, va_d, te_d = load_data() # инициализация наборов данных в формате MNIST

    training_inputs = [np.reshape(x, (784, 1)) for x in tr_d[0]] # преобразование массивов размера 1 на 784 к массивам размера 784 на 1

    training_results = [vectorized_result(y) for y in tr_d[1]] # представление цифр от 0 до 9 в виде массивов размера 10 на 1

    training_data = zip(training_inputs, training_results) # формируем набор обучающих данных из пар (x, y)

    validation_inputs = [np.reshape(x, (784, 1)) for x in va_d[0]] # преобразование массивов размера 1 на 784 к массивам размера 784 на 1

    validation_data = zip(validation_inputs, va_d[1]) # формируем набор данных проверки из пар (x, y)
```

Данная функция преобразует `training_data` в список, содержащий 50 000 пар (x, y), где x является 784-мерным numpy-массивом, содержащим входное изображение, а y – это 10-мерный numpy-массив, представляющий собой вектор, у которого координата с порядковым номером, соответствующим цифре на изображении, равняется единице, а остальные координаты нулевые. Аналогичные преобразования делаются для наборов `validation_data` и `test_data`.

Для преобразования числа в вектор-столбец (10-мерный numpy массив), используется следующая функция `vectorized_result`. Скопируйте ее программный код в файл `mnist_loader`.

```
def vectorized_result(j):
    e = np.zeros((10, 1)) e[j] = 1.0
    return e
```

□ Сохраните и закройте файл mnist_loader.py.

Запуск программы

В среде разработки IDLE последовательно выполните следующие команды для установки рабочего каталога на примере `C:\NeuralNetwork`:

```
>>>import os
>>>os.chdir('C:\\NeuralNetwork\\Network1')
```

Следующие команды используются для подключения модуля mnist_loader и инициализации наборов данных для обучения нейронной сети:

```
>>>import mnist_loader
>>>training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
```

Подключите созданный Вами модуль `network.py`:

```
>>>import network
```

При этом выполниться написанная в нем программа, выводящая информацию о нейронной сети.

Создайте нейронную сеть для распознавания рукописных цифр:

```
>>>net = network.Network([784, 30, 10])
```

Параметры, указанные при вызове данного метода, определяют топологию создаваемой сети. Таким образом, в результате выполнения команды будет создана сеть, состоящая из трех слоев: входной слой сети состоит из 784-х нейронов; внутренний слой из 30 нейронов и выходной слой из 10 нейронов.

Запустите процедуру обучения созданной нейронной сети, включающую 30 эпох:

```
>>>net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

Параметры, указанные при вызове метода `SGD`: обучающая выборка, количество эпох обучения, размер подвыборки, скорость обучения, тестирующая выборка.

Обучение может занять несколько минут. В ходе обучения будет выдаваться информация о пройденных эпохах (см. рис. 2). Для каждой эпохи выводится отношение количества правильно распознанных цифр к общему количеству цифр в тестовой выборке. Например, запись `Epoch 6: 9374 / 10000` говорит о том, что в результате эпохи обучения с номером 6 достигнута точность распознавания $\frac{9374}{10000} \approx 0.94$, что составляет 94%.

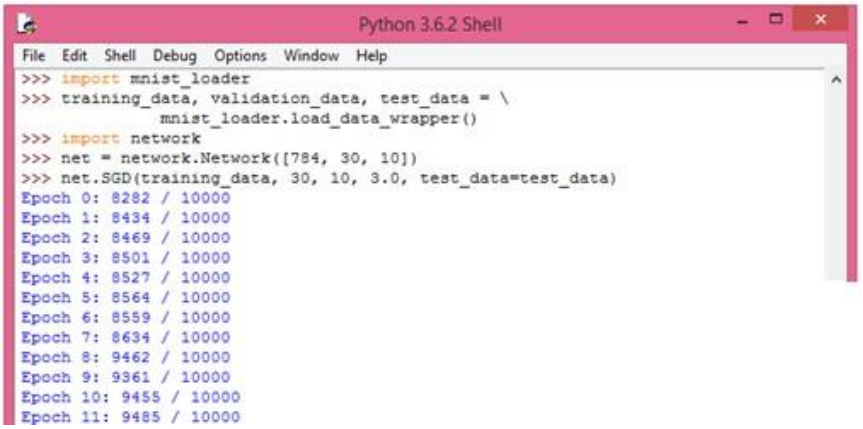


Рис.2. Результат работы программы Network1.

Цель работы: Написать компьютерную программу на языке Python 3 создающую и обучающую нейронную сеть для распознавания рукописных цифр с использованием метода градиентного спуска и базы данных MNIST

Рабочее задание

Выполнить инструкции приведенные выше

Контрольные вопросы

1.Какая архитектура нейронной сети обычно используется для задачи распознавания рукописных цифр, и какие типы слоев включаются в эту архитектуру?
2.Какой набор данных часто применяется при обучении нейронной сети для распознавания рукописных цифр, и как происходит подготовка и предобработка этого набора данных?
3.Как обучаются веса нейронной сети для распознавания рукописных цифр с использованием алгоритмов обратного распространения ошибки и градиентного спуска?
4.Как происходит процесс тестирования нейронной сети для распознавания рукописных цифр, и как оценивается ее точность и эффективность?
5.Какие вызовы и методы улучшения могут возникнуть при разработке и расширении нейронной сети для распознавания рукописных цифр, например, улучшение точности, ускорение обучения или обобщающая способность модели?

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Практическое занятие № 14.

Применение искусственного интеллекта при обучении модели.

Цель работы: Моделирование конфигурации нейронной сети и параметров скорости еѐ обучения.

Рабочее задание

Используя написанную ранее программу для распознавания рукописных цифр, создайте и обучите несколько нейронных сетей. Создаваемые сети должны иметь разную топологию. Для каждой сети попытайтесь подобрать оптимальные параметры для запуска процедуры обучения методом градиентного спуска.

Примеры запусков:

```
>>> net = network.Network([784, 100, 10])
>>> net.SGD(training_data, 30, 10, 3.0, test_data=test_data)


>>> net = network.Network([784, 100, 10])
>>> net.SGD(training_data, 30, 1000, 0.001, test_data=test_data)


>>> net = network.Network([784, 30, 10])
>>> net.SGD(training_data, 30, 10, 100.0, test_data=test_data)
```

Контрольные вопросы

1.Какие параметры влияют на обучение нейронной сети? Объясните характер их влияния?
2.Какова максимальная точность распознавания, которую вам удалось достичь при обучении нейронной сети (с указанием топологии нейронной сети)?
3.Как повлияло изменение топологии нейронной сети на качество ее обучения?
4.Как повлияло изменение параметров запуска метода градиентного спуска на качество обучения нейронной сети?
5. Основные показатели оценки результата и их критерии

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета:

- титульный лист;
- введение (цель и задачи);
- выполнение
- заключение

Время работы: 2 часа.

Перечень использованных информационных ресурсов

№	Автор	Название	Издательство	Гриф издания	Год издания	Кол-во в библиотеке	Наличие на электронных носителях	Электронные уч. пособия
1	2	3	4	5	6	7	8	9
3.2.1 Основная литература								
3.2.1.1	.Кук Д.	Машинное обучение с использованием библиотеки H2O	Москва: ДМК Пресс,		2022	-	-	https://www.studentlibrary.ru/book/ISBN9785970605080.html
3.2.2 Дополнительная литература								
3.2.2.1	Масленникова, О. Е.	Основы искусственного интеллекта	2-е изд., стер. - М.: ФЛИНТА		2023	-	-	http://znanium.com/catalog/product/465912
3.2.2.2	.Смолин Д.В.	Введение в искусственный интеллект	2-е изд., перераб. - М. : ФИЗМАТЛИТ		2022	-	-	http://www.studentlibrary.ru/book/ISBN9785922108621.html
	А.Л. Ездаков	Экспертные системы САПР: учебное пособие	М.: ИД ФОРУМ		2022			http://znanium.com/bookread.php?book=343778