МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ
имени И.Т. ТРУБИЛИНА»

Факультет прикладной информатики
Компьютерных технологий и систем

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ (МОДУЛЯ) «НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ»

Уровень высшего образования: магистратура

Направление подготовки: 09.04.02 Информационные системы и технологии

Направленность (профиль)подготовки: Проектно-исследовательская деятельность в области информационных технологий

Квалификация (степень) выпускника: магистр

Формы обучения: очная, заочная

Год набора: 2024

Срок получения образования: Очная форма обучения – 2 года
Заочная форма обучения – 2 года 5 месяца(-ев)

Объем: в зачетных единицах: 2 з.е.
в академических часах: 72 ак.ч.

2024

Разработчики:
Профессор, кафедра компьютерных технологий и систем
Луценко Е.В.

Рабочая программа дисциплины (модуля) составлена в соответствии с требованиями ФГОС ВО по направлению подготовки Направление подготовки: 09.04.02 Информационные системы и технологии, утвержденного приказом Минобрнауки России от 19.09.2017 №917, с учетом трудовых функций профессиональных стандартов: "Специалист по дизайну графических пользовательских интерфейсов", утвержден приказом Минтруда России от 29.09.2020 № 671н; "Руководитель разработки программного обеспечения", утвержден приказом Минтруда России от 20.07.2022 № 423н; "Системный аналитик", утвержден приказом Минтруда России от 27.04.2023 № 367н; "Системный администратор информационно-коммуникационных систем", утвержден приказом Минтруда России от 29.09.2020 № 680н; "Специалист по научно-исследовательским и опытно-конструкторским разработкам", утвержден приказом Минтруда России от 04.03.2014 № 121н.

Согласование и утверждение

№	Подразделение или коллегиальный орган	Ответственное лицо	ФИО	Виза	Дата, протокол (при наличии)
1	Компьютерных технологий и систем	Заведующий кафедрой, руководитель подразделения, реализующего ОП	Лукьяненко Т.В.	Согласовано	22.03.2024, № 9

1. Цель и задачи освоения дисциплины (модуля)

Цель освоения дисциплины - заключается в подготовке студентов к успешной работе с современными методами и инструментами нейронных сетей, расширении их технических и аналитических навыков, а также стимулировании творческого подхода к решению сложных задач в области информационных технологий.

Задачи изучения дисциплины:

- Изучение принципов работы и структурных компонентов искусственных нейронных сетей.;
- Освоение различных алгоритмов обучения нейронных сетей и их применение для решения задач классификации, регрессии, кластеризации и обработки данных.;
- Исследование продвинутых архитектур нейронных сетей, таких как сверточные нейронные сети, рекуррентные нейронные сети и глубокие нейронные сети;
- Рассмотрение прикладных областей использования нейронных сетей, таких как компьютерное зрение, обработка естественного языка, обнаружение аномалий и многие другие.¶;
- Ознакомление с инструментами и библиотеками программирования для создания, обучения и оценки нейронных сетей..

2. Планируемые результаты обучения по дисциплине (модулю), соотнесенные с планируемыми результатами освоения образовательной программы

Компетенции, индикаторы и результаты обучения

ПК-П1 Способен разрабатывать и исследовать модели объектов профессиональной деятельности, предлагать и адаптировать методики, определять качество проводимых исследований, составлять отчеты о проделанной работе, обзоры, готовить публикации.

ПК-П1.1 Координация деятельности соисполнителей, участвующих в выполнении работ с другими организациями

Знать:

ПК-П1.1/Зн1 Отечественная и международная нормативная база в соответствующей области знаний

ПК-П1.1/Зн2 Научная проблематика соответствующей области знаний

ПК-П1.1/Зн3 Методы, средства и практика планирования, организации, проведения и внедрения научных исследований и опытно-конструкторских разработок

Уметь:

ПК-П1.1/Ум1 Применять актуальную нормативную документацию в соответствующей области знаний

ПК-П1.1/Ум2 Анализировать научную проблематику соответствующей области знаний

ПК-П1.1/Ум3 Применять методы и средства планирования, организации, проведения и внедрения научных исследований и опытно-конструкторских разработок

Владеть:

ПК-П1.1/Нв1 Анализ результатов работ соисполнителей, участвующих в выполнении работ с другими организациями

ПК-П1.1/Нв2 Разработка мероприятий по координации деятельности соисполнителей, участвующих в выполнении работ с другими организациями

ПК-П1.1/Нв3 Контроль реализации планов мероприятий по координации деятельности соисполнителей, участвующих в выполнении работ с другими организациями

ПК-П1.1/Нв4 Подготовка и представление руководству отчетов о реализации планов мероприятий по координации деятельности соисполнителей, участвующих в выполнении работ с другими организациями

ПК-П1.2 Определение сферы применения результатов научно-исследовательских и опытно-конструкторских работ

Знать:

ПК-П1.2/Зн1 Отечественная и международная нормативная база в соответствующей области знаний

ПК-П1.2/Зн2 Основы экономики, организации производства, труда и управления организацией

ПК-П1.2/Зн3 Методы разработки информационных, объектных, документных моделей производственных организаций

Уметь:

ПК-П1.2/Ум1 Применять актуальную нормативную документацию в соответствующей области знаний

ПК-П1.2/Ум2 Применять методы разработки информационных, объектных, документных моделей производственных предприятий

Владеть:

ПК-П1.2/Нв1 Анализ возможных областей применения результатов научно-исследовательских и опытно-конструкторских работ

ПК-П1.2/Нв2 Организация внедрения результатов научно-исследовательских и опытно-конструкторских работ

ПК-П1.2/Нв3 Обеспечение научного руководства практической реализацией результатов научных исследований и опытно- конструкторских работ

ПК-П1.2/Нв4 Контроль реализации внедрения результатов научно-исследовательских и опытно-конструкторских работ

ПК-П1.2/Нв5 Осуществление подготовки и представления руководству отчета о практической реализации результатов научных исследований и опытно-конструкторских работ

ПК-П1.3 Владеет навыками формирования новых направлений научных исследований и опытно-конструкторских разработок

Знать:

ПК-П1.3/Зн1 Отечественная и международная нормативная база в соответствующей области знаний

ПК-П1.3/Зн2 Научная проблематика соответствующей области знаний

ПК-П1.3/Зн3 Методы, средства и практика планирования, организации, проведения и внедрения научных исследований и опытно-конструкторских разработок

Уметь:

ПК-П1.3/Ум1 Применять актуальную нормативную документацию в соответствующей области знаний

ПК-П1.3/Ум2 Анализировать новую научную проблематику соответствующей области знаний

ПК-П1.3/Ум3 Применять методы и средства планирования, организации, проведения и внедрения научных исследований и опытно-конструкторских разработок

Владеть:

ПК-П1.3/Нв1 Проведение анализа новых направлений исследований в соответствующей области знаний

ПК-П1.3/Нв2 Обоснование перспектив проведения исследований в соответствующей области знаний

ПК-П1.3/Нв3 Формирование программ проведения исследований в новых направлениях

ПК-П12 Способен вести сдачу проекта, собирать и анализировать мнения и замечания заказчика по выполнению проекта и предлагать соответствующие решения.

ПК-П12.1 Знает методы планирования и организации работ подчиненных системных аналитиков на всем жизненном цикле системы

Знать:

ПК-П12.1/Зн1 Методы календарно-ресурсного планирования

ПК-П12.1/Зн2 Методы и инструменты обследования, проектирования и разработки требований и проектных решений

ПК-П12.1/Зн3 Технология производства программного обеспечения

ПК-П12.1/Зн4 Общие понятия о функциях потребителей требований и проектных решений: тестировщиков, программистов, архитекторов, технических писателей, администраторов, специалистов технической поддержки

Уметь:

ПК-П12.1/Ум1 Пользоваться инструментами календарно-ресурсного планирования

ПК-П12.1/Ум2 Пользоваться системами управления задачами

ПК-П12.1/Ум3 Вести деловые переговоры и конфликтные переговоры

ПК-П12.1/Ум4 Фасилитировать и модерировать работу группы

ПК-П12.1/Ум5 Вести деловую переписку

ПК-П12.1/Ум6 Формализовывать входящие требования и запросы

ПК-П12.1/Ум7 Организовывать проектные работы

ПК-П12.1/Ум8 Управлять работой группы

Владеть:

ПК-П12.1/Нв1 Выявление потребителей, целей и контекста использования требований и проектных решений

ПК-П12.1/Нв2 Определение источников информации для требований и проектных решений

ПК-П12.1/Нв3 Выбор методов разработки требований и проектных решений

ПК-П12.1/Нв4 Выбор типов и атрибутов требований и элементов проектных решений

ПК-П12.1/Нв5 Выбор шаблонов промежуточных и финальных документов для требований и проектных решений

ПК-П12.1/Нв6 Составление и согласование перечня поставок

ПК-П12.1/Нв7 Достижение договоренностей с потребителями требований и проектных решений о методах и процедуре приемки требований

ПК-П12.1/Нв8 Определение состава работ по разработке требований

ПК-П12.1/Нв9 Создание календарно-ресурсного графика работ

ПК-П12.1/Нв10 Определение требований к компетенциям исполнителей разных работ по созданию требований

ПК-П12.1/Нв11 Определение графика контрольных мероприятий по аналитическим работам

ПК-П12.1/Нв12 Определение кандидатов на исполнение отдельных аналитических работ

ПК-П12.1/Нв13 Постановка задач на разработку планов аналитических работ по отдельным частям системы

ПК-П12.1/Нв14 Интеграция планов аналитических работ по отдельным частям системы в единый план

ПК-П12.1/Нв15 Согласование плана аналитических работ с менеджером проекта

ПК-П12.1/Нв16 Определение состава аналитической группы проекта

ПК-П12.1/Нв17 Проведение знакомства участников аналитической группы

ПК-П12.1/Нв18 Представление и обсуждение плана аналитических работ

ПК-П12.1/Нв19 Распределение ролей и аналитических работ между участниками аналитической группы проекта

ПК-П12.1/Нв20 Ответы на вопросы и предложения участников аналитической группы проекта

ПК-П12.1/Нв21 Достижение соглашений с владельцами ресурсов о выделении ресурсов для выполнения аналитических работ в проекте

ПК-П12.2 Умеет планироватье и организовывать работы подчиненных системных аналитиков на всем жизненном цикле системы

Знать:

ПК-П12.2/Зн1 Методы календарно-ресурсного планирования

ПК-П12.2/Зн2 Методы и инструменты обследования, проектирования и разработки требований и проектных решений

ПК-П12.2/Зн3 Виды документов и артефактов требований и проектных решений

ПК-П12.2/Зн4 Технология построения автоматизированных систем

ПК-П12.2/Зн5 Технология производства программного обеспечения

ПК-П12.2/Зн6 Общие понятия о функциях потребителей требований и проектных решений: тестировщиков, программистов, архитекторов, технических писателей, администраторов, специалистов технической поддержки

Уметь:

ПК-П12.2/Ум1 Пользоваться инструментами календарно-ресурсного планирования

ПК-П12.2/Ум2 Пользоваться системами управления задачами

ПК-П12.2/Ум3 Вести деловые переговоры и конфликтные переговоры

ПК-П12.2/Ум4 Фасилитировать и модерировать работу группы

ПК-П12.2/Ум5 Формализовывать входящие требования и запросы

ПК-П12.2/Ум6 Организовывать проектные работы

ПК-П12.2/Ум7 Проводить совещания

ПК-П12.2/Ум8 Управлять работой группы

Владеть:

ПК-П12.2/Нв1 Выявление потребителей, целей и контекста использования требований и проектных решений

ПК-П12.2/Нв2 Определение источников информации для требований и проектных решений

ПК-П12.2/Нв3 Выбор методов разработки требований и проектных решений

ПК-П12.2/Нв4 Выбор типов и атрибутов требований и элементов проектных решений

ПК-П12.2/Нв5 Выбор шаблонов промежуточных и финальных документов для требований и проектных решений

ПК-П12.2/Нв6 Достижение договоренностей с потребителями требований и проектных решений о методах и процедуре приемки требований

ПК-П12.2/Нв7 Определение состава работ по разработке требований

ПК-П12.2/Нв8 Создание календарно-ресурсного графика работ

ПК-П12.2/Нв9 Определение требований к компетенциям исполнителей разных работ по созданию требований

ПК-П12.2/Нв10 Определение графика контрольных мероприятий по аналитическим работам

ПК-П12.2/Нв11 Определение кандидатов на исполнение отдельных аналитических работ

ПК-П12.2/Нв12 Постановка задач на разработку планов аналитических работ по отдельным частям системы

ПК-П12.2/Нв13 Интеграция планов аналитических работ по отдельным частям системы в единый план

ПК-П12.2/Нв14 Согласование плана аналитических работ с менеджером проекта

ПК-П12.2/Нв15 Определение состава аналитической группы проекта

ПК-П12.2/Нв16 Проведение знакомства участников аналитической группы

ПК-П12.2/Нв17 Представление и обсуждение плана аналитических работ

ПК-П12.2/Нв18 Распределение ролей и аналитических работ между участниками аналитической группы проекта

ПК-П12.2/Нв19 Ответы на вопросы и предложения участников аналитической группы проекта

ПК-П12.2/Нв20 Достижение соглашений с владельцами ресурсов о выделении ресурсов для выполнения аналитических работ в проекте

ПК-П12.3 Владеет навыками планирования и организации работ подчиненных системных аналитиков на всем жизненном цикле системы

Знать:

ПК-П12.3/Зн1 Методы календарно-ресурсного планирования

ПК-П12.3/Зн2 Методы и инструменты обследования, проектирования и разработки требований и проектных решений

ПК-П12.3/Зн3 Виды документов и артефактов требований и проектных решений

ПК-П12.3/Зн4 Технология производства программного обеспечения

ПК-П12.3/Зн5 Общие понятия о функциях потребителей требований и проектных решений: тестировщиков, программистов, архитекторов, технических писателей, администраторов, специалистов технической поддержки

Уметь:

ПК-П12.3/Ум1 Пользоваться инструментами календарно-ресурсного планирования

ПК-П12.3/Ум2 Пользоваться системами управления задачами

ПК-П12.3/Ум3 Вести деловые переговоры и конфликтные переговоры

ПК-П12.3/Ум4 Фасилитировать и модерировать работу группы

ПК-П12.3/Ум5 Вести деловую переписку

ПК-П12.3/Ум6 Формализовывать входящие требования и запросы

ПК-П12.3/Ум7 Организовывать проектные работы

ПК-П12.3/Ум8 Проводить совещания

ПК-П12.3/Ум9 Управлять работой группы

Владеть:

ПК-П12.3/Нв1 Выявление потребителей, целей и контекста использования требований и проектных решений

ПК-П12.3/Нв2 Определение источников информации для требований и проектных решений

ПК-П12.3/Нв3 Выбор методов разработки требований и проектных решений

ПК-П12.3/Нв4 Выбор типов и атрибутов требований и элементов проектных решений

ПК-П12.3/Нв5 Выбор шаблонов промежуточных и финальных документов для требований и проектных решений

ПК-П12.3/Нв6 Составление и согласование перечня поставок

ПК-П12.3/Нв7 Достижение договоренностей с потребителями требований и проектных решений о методах и процедуре приемки требований

ПК-П12.3/Нв8 Определение состава работ по разработке требований

ПК-П12.3/Нв9 Создание календарно-ресурсного графика работ

ПК-П12.3/Нв10 Определение требований к компетенциям исполнителей разных работ по созданию требований

ПК-П12.3/Нв11 Определение графика контрольных мероприятий по аналитическим работам

ПК-П12.3/Нв12 Определение кандидатов на исполнение отдельных аналитических работ

ПК-П12.3/Нв13 Постановка задач на разработку планов аналитических работ по отдельным частям системы

ПК-П12.3/Нв14 Интеграция планов аналитических работ по отдельным частям системы в единый план

ПК-П12.3/Нв15 Согласование плана аналитических работ с менеджером проекта

ПК-П12.3/Нв16 Определение состава аналитической группы проекта

ПК-П12.3/Нв17 Представление и обсуждение плана аналитических работ

ПК-П12.3/Нв18 Распределение ролей и аналитических работ между участниками аналитической группы проекта

ПК-П12.3/Нв19 Ответы на вопросы и предложения участников аналитической группы проекта

ПК-П12.3/Нв20 Достижение соглашений с владельцами ресурсов о выделении ресурсов для выполнения аналитических работ в проекте

ПК-П15 Способность использовать знание основных методов искусственного интеллекта в последующей профессиональной деятельности в качестве научных сотрудников, преподавателей образовательных организаций высшего образования, инженеров, технологов.

ПК-П15.1 Знает методы управления информацией в процессе разработки компьютерного программного обеспечения

Знать:

ПК-П15.1/Зн1 Методологии разработки компьютерного программного обеспечения

ПК-П15.1/Зн2 Методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.1/Зн3 Методологии организации системы управления версиями, репозитория, системы учета задач и дефектов, системы сборки и непрерывной интеграции, базы знаний для разработки компьютерного программного обеспечения

ПК-П15.1/Зн4 Лучшие практики управления разработкой компьютерного программного обеспечения

ПК-П15.1/Зн5 Основные принципы и методы управления персоналом

ПК-П15.1/Зн6 Нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

ПК-П15.1/Зн7 Технологии межличностной и групповой коммуникации в деловом взаимодействии, основы конфликтологии

Уметь:

ПК-П15.1/Ум1 Применять методологии разработки компьютерного программного обеспечения

ПК-П15.1/Ум2 Применять методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.1/Ум3 Применять лучшие практики разработки компьютерного программного обеспечения и отражать их в базе знаний

ПК-П15.1/Ум4 Применять основные принципы и методы управления персоналом

ПК-П15.1/Ум5 Применять нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

Владеть:

ПК-П15.1/Нв1 Разработка регламентов обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.1/Нв2 Мониторинг соблюдения регламента обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.1/Нв3 Принятие управленческих решений по результатам мониторинга соблюдения регламента обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.2 Умеет управлять информацией в процессе разработки компьютерного программного обеспечения

Знать:

ПК-П15.2/Зн1 Методологии разработки компьютерного программного обеспечения

ПК-П15.2/Зн2 Методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.2/Зн3 Методологии организации системы управления версиями, репозитория, системы учета задач и дефектов, системы сборки и непрерывной интеграции, базы знаний для разработки компьютерного программного обеспечения

ПК-П15.2/Зн4 Лучшие практики управления разработкой компьютерного программного обеспечения

ПК-П15.2/Зн5 Основные принципы и методы управления персоналом

ПК-П15.2/Зн6 Нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

Уметь:

ПК-П15.2/Ум1 Применять методологии разработки компьютерного программного обеспечения

ПК-П15.2/Ум2 Применять методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.2/Ум3 Применять лучшие практики разработки компьютерного программного обеспечения и отражать их в базе знаний

ПК-П15.2/Ум4 Применять нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

ПК-П15.2/Ум5 Осуществлять коммуникации с заинтересованными сторонами

Владеть:

ПК-П15.2/Нв1 Организация системы контроля версий, репозитория, системы учета задач и дефектов, системы сборки и непрерывной интеграции, базы знаний для разработки компьютерного программного обеспечения

ПК-П15.2/Нв2 Разработка регламентов обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.2/Нв3 Мониторинг соблюдения регламента обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.3 Владеет навыками управления информацией в процессе разработки компьютерного программного обеспечения

Знать:

ПК-П15.3/Зн1 Методологии разработки компьютерного программного обеспечения

ПК-П15.3/Зн2 Методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.3/Зн3 Методологии организации системы управления версиями, репозитория, системы учета задач и дефектов, системы сборки и непрерывной интеграции, базы знаний для разработки компьютерного программного обеспечения

ПК-П15.3/Зн4 Лучшие практики управления разработкой компьютерного программного обеспечения

ПК-П15.3/Зн5 Нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

ПК-П15.3/Зн6 Технологии межличностной и групповой коммуникации в деловом взаимодействии, основы конфликтологии

Уметь:

ПК-П15.3/Ум1 Применять методологии разработки компьютерного программного обеспечения

ПК-П15.3/Ум2 Применять методологии управления проектами разработки компьютерного программного обеспечения

ПК-П15.3/Ум3 Применять лучшие практики разработки компьютерного программного обеспечения и отражать их в базе знаний

ПК-П15.3/Ум4 Применять основные принципы и методы управления персоналом

ПК-П15.3/Ум5 Применять нормативно-технические документы (стандарты и регламенты), описывающие процессы управления информацией в команде разработки компьютерного программного обеспечения

ПК-П15.3/Ум6 Осуществлять коммуникации с заинтересованными сторонами

Владеть:

ПК-П15.3/Нв1 Организация системы контроля версий, репозитория, системы учета задач и дефектов, системы сборки и непрерывной интеграции, базы знаний для разработки компьютерного программного обеспечения

ПК-П15.3/Нв2 Разработка регламентов обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.3/Нв3 Мониторинг соблюдения регламента обмена информацией в команде разработчиков компьютерного программного обеспечения

ПК-П15.3/Нв4 Принятие управленческих решений по результатам мониторинга соблюдения регламента обмена информацией в команде разработчиков компьютерного программного обеспечения

3. Место дисциплины в структуре ОП

Дисциплина (модуль) «Нейросетевые технологии в информационных системах» относится к формируемой участниками образовательных отношений части образовательной программы и изучается в семестре(ах): Очная форма обучения - 3, Заочная форма обучения - 4.

В процессе изучения дисциплины студент готовится к видам профессиональной деятельности и решению профессиональных задач, предусмотренных ФГОС ВО и образовательной программой.

4. Объем дисциплины и виды учебной работы

Очная форма обучения

Период обучения	Общая трудоемкость (часы)	Общая трудоемкость (ЗЕТ)	Контактная работа (часы, всего)	Внеаудиторная контактная работа (часы)	Зачет (часы)	Лекционные занятия (часы)	Практические занятия (часы)	Самостоятельная работа (часы)	Промежуточная аттестация (часы)
Третий семестр	72	2	37	1		18	18	35	Зачет
Всего	72	2	37	1		18	18	35	

Заочная форма обучения

Период обучения	Общая трудоемкость (часы)	Общая трудоемкость (ЗЕТ)	Контактная работа (часы, всего)	Внеаудиторная контактная работа (часы)	Зачет (часы)	Лекционные занятия (часы)	Практические занятия (часы)	Самостоятельная работа (часы)	Промежуточная аттестация (часы)
Четвертый семестр	72	2	13	1	4	2	6	59	Зачет (4) Контрольная работа
Всего	72	2	13	1	4	2	6	59	

5. Содержание дисциплины

5.1. Разделы, темы дисциплины и виды занятий

(часы промежуточной аттестации не указываются)

Очная форма обучения

Наименование раздела, темы	Всего	Внеаудиторная контактная работа	Лекционные занятия	Практические занятия	Самостоятельная работа	Планируемые результаты обучения, соотнесенные с результатами освоения программы
Раздел 1. НЕЙРОНЫ И ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ	**2,4**	**1**	**0,4**	**0,4**	**0,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3

Тема 1.1. Комбинирование входных сигналов	1,7	1	0,2	0,2	0,3	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 1.2. Функция активации элемента.	0,7		0,2	0,2	0,3	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 2. ИСТОРИЯ НЕЙРОННЫХ СЕТЕЙ	**1,4**		**0,4**	**0,4**	**0,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 2.1. Работы Дональда Олдинга Хебба, Уоррена Мак-Каллока и Уолтера Питтса	0,7		0,2	0,2	0,3	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 2.2. Работы Френка Розенблатта	0,7		0,2	0,2	0,3	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 3. КЛАССИФИКАЦИЯ НЕЙРОННЫХ СЕТЕЙ	**2,1**		**0,6**	**0,6**	**0,9**	ПК-П1.1 ПК-П1.2
Тема 3.1. Классификация нейронных сетей по характеру обучения	0,7		0,2	0,2	0,3	ПК-П1.3 ПК-П12.1 ПК-П12.2
Тема 3.2. Классификация нейронных сетей по типу настройки весов	0,7		0,2	0,2	0,3	ПК-П12.3 ПК-П15.1 ПК-П15.2
Тема 3.3. Классификация нейронных сетей по типу входной информации	0,7		0,2	0,2	0,3	ПК-П15.3
Раздел 4. АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ	**1,4**		**0,4**	**0,4**	**0,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 4.1. Полносвязные нейронные сети	0,7		0,2	0,2	0,3	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 4.2. Многослойные нейронные сети	0,7		0,2	0,2	0,3	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 5. ФОРМАЛЬНЫЙ НЕЙРОН	**2,3**		**0,6**	**0,6**	**1,1**	ПК-П1.1 ПК-П1.2
Тема 5.1. Параметры нейрона	0,7		0,2	0,2	0,3	ПК-П1.3 ПК-П12.1 ПК-П12.2
Тема 5.2. Функция активации нейрона	0,8		0,2	0,2	0,4	ПК-П12.3 ПК-П15.1
Тема 5.3. Недостатки формального нейрона	0,8		0,2	0,2	0,4	ПК-П15.2 ПК-П15.3
Раздел 6. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ	**1,6**		**0,4**	**0,4**	**0,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 6.1. Сеть из одного нейрона	0,8		0,2	0,2	0,4	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 6.2. Однослойный перцептрон Френка Розенблатта	0,8		0,2	0,2	0,4	ПК-П15.1 ПК-П15.2 ПК-П15.3

Раздел 7. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ	**3,2**		**0,8**	**0,8**	**1,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 7.1. Что такое обучение нейронной сети	0,8		0,2	0,2	0,4	
Тема 7.2. Алгоритмы обучения нейронной сети с учителем и без учителя	1,2		0,3	0,3	0,6	
Тема 7.3. Итерационные алгоритмы обучения нейронных сетей	1,2		0,3	0,3	0,6	
Раздел 8. МНОГОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ	**2,4**		**0,6**	**0,6**	**1,2**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 8.1. Алгоритм обратного распространения ошибки	1,2		0,3	0,3	0,6	
Тема 8.2. Алгоритм обучения многослойной НС	1,2		0,3	0,3	0,6	
Раздел 9. ВВЕДЕНИЕ В KERAS И ЕГО ОСНОВНЫЕ ПРИНЦИПЫ	**7,2**		**1,8**	**1,8**	**3,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 9.1. Что такое Keras и глубинное обучение?	1,2		0,3	0,3	0,6	
Тема 9.2. Методы глубинного обучения	1,2		0,3	0,3	0,6	
Тема 9.3. Важность глубинного обучения	1,2		0,3	0,3	0,6	
Тема 9.4. Микросервисы глубинного изучения	1,2		0,3	0,3	0,6	
Тема 9.5. Open Source фреймворки о глубинном обучении	1,2		0,3	0,3	0,6	
Тема 9.6. Основные принципы Keras	1,2		0,3	0,3	0,6	
Раздел 10. МОДЕЛИ KERAS	**4,8**		**1,2**	**1,2**	**2,4**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 10.1. Основная структура данных Keras	1,2		0,3	0,3	0,6	
Тема 10.2. Краткая характеристка моделей Keras	1,2		0,3	0,3	0,6	
Тема 10.3. API класса Model	1,2		0,3	0,3	0,6	
Тема 10.4. Основные методы класса Model	1,2		0,3	0,3	0,6	
Раздел 11. СЛОИ В KERAS	**4,8**		**1,2**	**1,2**	**2,4**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 11.1. Общие методы слоев	1,2		0,3	0,3	0,6	
Тема 11.2. Плотный слой Dense	1,2		0,3	0,3	0,6	
Тема 11.3. Сверточные слои	1,2		0,3	0,3	0,6	
Тема 11.4. Слой пулинга	1,2		0,3	0,3	0,6	

Раздел 12. ОСНОВЫ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОЙ МОДЕЛЬЮ KERAS	**6**		**1,5**	**1,5**	**3**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 12.1. Создание последовательной модели (Sequential)	1,2		0,3	0,3	0,6	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 12.2. Указание размерности входных данных	1,2		0,3	0,3	0,6	ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 12.3. Компиляция	1,2		0,3	0,3	0,6	
Тема 12.4. Обучение	1,2		0,3	0,3	0,6	
Тема 12.5. Пример многослойного перцептрона (MLP) для многоклассовой классификации	1,2		0,3	0,3	0,6	
Раздел 13. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В KERAS	**3,6**		**0,9**	**0,9**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 13.1. Сверточная нейронная сеть	1,2		0,3	0,3	0,6	
Тема 13.2. Набор данных - CIFAR10	1,2		0,3	0,3	0,6	
Тема 13.3. Обучение сети	1,2		0,3	0,3	0,6	
Раздел 14. РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ	**4,8**		**1,2**	**1,2**	**2,4**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 14.1. Описание набора данных (датасет) MNIST	1,2		0,3	0,3	0,6	
Тема 14.2. Базовая модель с многослойным перцептроном	1,2		0,3	0,3	0,6	
Тема 14.3. Простая сверточная нейронная сеть для MNIST	1,2		0,3	0,3	0,6	
Тема 14.4. Большая сверточная нейронная сеть для MNIST	1,2		0,3	0,3	0,6	
Раздел 15. ПРЕДСТАВЛЕНИЕ СЛОВ В ВЕКТОРНОМ ПРОСТРАНСТВЕ	**3,6**		**0,9**	**0,9**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 15.1. Векторизация слов	1,2		0,3	0,3	0,6	ПК-П12.1 ПК-П12.2
Тема 15.2. Embedding слой Keras	1,2		0,3	0,3	0,6	ПК-П12.3 ПК-П15.1
Тема 15.3. Пример обучения векторизации	1,2		0,3	0,3	0,6	ПК-П15.2 ПК-П15.3
Раздел 16. LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ	**8,4**		**2,1**	**2,1**	**4,2**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1

Тема 16.1. Рекуррентные нейронные сети	1,2		0,3	0,3	0,6	ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 16.2. Полностью рекуррентная сеть	1,2		0,3	0,3	0,6	
Тема 16.3. Проблема долгосрочных зависимостей	1,2		0,3	0,3	0,6	
Тема 16.4. LSTM сети	1,2		0,3	0,3	0,6	
Тема 16.5. Главная идея LSTM	1,2		0,3	0,3	0,6	
Тема 16.6. Разновидности LSTM сетей	1,2		0,3	0,3	0,6	
Тема 16.7. Прогнозирование временных рядов	1,2		0,3	0,3	0,6	
Раздел 17. КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ	**2,4**		**0,6**	**0,6**	**1,2**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 17.1. Проблема классификации последовательностей, ее сложность и решаемая задача	1,2		0,3	0,3	0,6	
Тема 17.2. Описание набора данных и решения задачи	1,2		0,3	0,3	0,6	
Раздел 18. НЕЙРОННЫЕ СЕТИ НА ОСНОВЕ БИБЛИОТЕКИ TENSORFLOW	**9,6**		**2,4**	**2,4**	**4,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 18.1. Начало работы с TensorFlow	1,2		0,3	0,3	0,6	
Тема 18.2. Основы работы в TensorFlow	1,2		0,3	0,3	0,6	
Тема 18.3. Определение вычислительных графов в TensorFlow	1,2		0,3	0,3	0,6	
Тема 18.4. Визуализация вычислительного графа с помощью TensorBoard	1,2		0,3	0,3	0,6	
Тема 18.5. Математика с TensorFlow	1,2		0,3	0,3	0,6	
Тема 18.6. Тензорные операции	1,2		0,3	0,3	0,6	
Тема 18.7. Матричные операции	1,2		0,3	0,3	0,6	
Тема 18.8. Пример нейронной сети в TensorFlow	1,2		0,3	0,3	0,6	
Итого	**72**	**1**	**18**	**18**	**35**	

Заочная форма обучения

Наименование раздела, темы		ная контактная абота	е занятия	кие занятия	льная работа	ые результаты оотнесенные с ии освоения

	Всего	Внеаудиторная	Лекционные	Практическ	Самостоятє	Планируемы обучения, с результатам программы
Раздел 1. НЕЙРОНЫ И ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ	**3,54**	**1**	**0,04**	**0,2**	**2,3**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 1.1. Комбинирование входных сигналов	2,32	1	0,02	0,1	1,2	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 1.2. Функция активации элемента.	1,22		0,02	0,1	1,1	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 2. ИСТОРИЯ НЕЙРОННЫХ СЕТЕЙ	**2,04**		**0,04**	**0,2**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 2.1. Работы Дональда Олдинга Хебба, Уоррена Мак-Каллока и Уолтера Питтса	1,02		0,02	0,1	0,9	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 2.2. Работы Френка Розенблатта	1,02		0,02	0,1	0,9	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 3. КЛАССИФИКАЦИЯ НЕЙРОННЫХ СЕТЕЙ	**3,06**		**0,06**	**0,3**	**2,7**	ПК-П1.1 ПК-П1.2
Тема 3.1. Классификация нейронных сетей по характеру обучения	1,02		0,02	0,1	0,9	ПК-П1.3 ПК-П12.1 ПК-П12.2
Тема 3.2. Классификация нейронных сетей по типу настройки весов	1,02		0,02	0,1	0,9	ПК-П12.3 ПК-П15.1 ПК-П15.2
Тема 3.3. Классификация нейронных сетей по типу входной информации	1,02		0,02	0,1	0,9	ПК-П15.3
Раздел 4. АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ	**2,04**		**0,04**	**0,2**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 4.1. Полносвязные нейронные сети	1,02		0,02	0,1	0,9	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 4.2. Многослойные нейронные сети	1,02		0,02	0,1	0,9	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 5. ФОРМАЛЬНЫЙ НЕЙРОН	**3,06**		**0,06**	**0,3**	**2,7**	ПК-П1.1 ПК-П1.2
Тема 5.1. Параметры нейрона	1,02		0,02	0,1	0,9	ПК-П1.3 ПК-П12.1 ПК-П12.2
Тема 5.2. Функция активации нейрона	1,02		0,02	0,1	0,9	ПК-П12.3 ПК-П15.1
Тема 5.3. Недостатки формального нейрона	1,02		0,02	0,1	0,9	ПК-П15.2 ПК-П15.3
Раздел 6. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ	**2,04**		**0,04**	**0,2**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3

Тема 6.1. Сеть из одного нейрона	1,02		0,02	0,1	0,9	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 6.2. Однослойный перцептрон Френка Розенблатта	1,02		0,02	0,1	0,9	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 7. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ	**3,04**		**0,06**	**0,28**	**2,7**	ПК-П1.1 ПК-П1.2
Тема 7.1. Что такое обучение нейронной сети	1,02		0,02	0,1	0,9	ПК-П1.3 ПК-П12.1
Тема 7.2. Алгоритмы обучения нейронной сети с учителем и без учителя	1,01		0,02	0,09	0,9	ПК-П12.2 ПК-П12.3 ПК-П15.1
Тема 7.3. Итерационные алгоритмы обучения нейронных сетей	1,01		0,02	0,09	0,9	ПК-П15.2 ПК-П15.3
Раздел 8. МНОГОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ	**2,02**		**0,04**	**0,18**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 8.1. Алгоритм обратного распространения ошибки	1,01		0,02	0,09	0,9	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 8.2. Алгоритм обучения многослойной НС	1,01		0,02	0,09	0,9	ПК-П15.1 ПК-П15.2 ПК-П15.3
Раздел 9. ВВЕДЕНИЕ В KERAS И ЕГО ОСНОВНЫЕ ПРИНЦИПЫ	**6,09**		**0,15**	**0,54**	**5,4**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 9.1. Что такое Keras и глубинное обучение?	1,01		0,02	0,09	0,9	ПК-П12.1 ПК-П12.2
Тема 9.2. Методы глубинного обучения	1,01		0,02	0,09	0,9	ПК-П12.3 ПК-П15.1
Тема 9.3. Важность глубинного обучения	1,01		0,02	0,09	0,9	ПК-П15.2 ПК-П15.3
Тема 9.4. Микросервисы глубинного изучения	1,02		0,03	0,09	0,9	
Тема 9.5. Open Source фреймворки о глубинном обучении	1,02		0,03	0,09	0,9	
Тема 9.6. Основные принципы Keras	1,02		0,03	0,09	0,9	
Раздел 10. МОДЕЛИ KERAS	**4,08**		**0,12**	**0,36**	**3,6**	ПК-П1.1 ПК-П1.2
Тема 10.1. Основная структура данных Keras	1,02		0,03	0,09	0,9	ПК-П1.3 ПК-П12.1
Тема 10.2. Краткая характеристка моделей Keras	1,02		0,03	0,09	0,9	ПК-П12.2 ПК-П12.3
Тема 10.3. API класса Model	1,02		0,03	0,09	0,9	ПК-П15.1
Тема 10.4. Основные методы класса Model	1,02		0,03	0,09	0,9	ПК-П15.2 ПК-П15.3
Раздел 11. СЛОИ В KERAS	**4,35**		**0,39**	**0,36**	**3,6**	ПК-П1.1 ПК-П1.2
Тема 11.1. Общие методы слоев	1,02		0,03	0,09	0,9	ПК-П1.3 ПК-П12.1

Тема 11.2. Плотный слой Dense	1,02		0,03	0,09	0,9	ПК-П12.1 ПК-П12.2
Тема 11.3. Сверточные слои	1,02		0,03	0,09	0,9	ПК-П12.3 ПК-П15.1
Тема 11.4. Слой пулинга	1,29		0,3	0,09	0,9	ПК-П15.2 ПК-П15.3
Раздел 12. ОСНОВЫ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОЙ МОДЕЛЬЮ KERAS	**5,1**		**0,15**	**0,45**	**4,5**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 12.1. Создание последовательной модели (Sequential)	1,02		0,03	0,09	0,9	ПК-П12.1 ПК-П12.2 ПК-П12.3
Тема 12.2. Указание размерности входных данных	1,02		0,03	0,09	0,9	ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 12.3. Компиляция	1,02		0,03	0,09	0,9	
Тема 12.4. Обучение	1,02		0,03	0,09	0,9	
Тема 12.5. Пример многослойного перцептрона (MLP) для многоклассовой классификации	1,02		0,03	0,09	0,9	
Раздел 13. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В KERAS	**3,06**		**0,09**	**0,27**	**2,7**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 13.1. Сверточная нейронная сеть	1,02		0,03	0,09	0,9	
Тема 13.2. Набор данных - CIFAR10	1,02		0,03	0,09	0,9	
Тема 13.3. Обучение сети	1,02		0,03	0,09	0,9	
Раздел 14. РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ	**4,08**		**0,12**	**0,36**	**3,6**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 14.1. Описание набора данных (датасет) MNIST	1,02		0,03	0,09	0,9	
Тема 14.2. Базовая модель с многослойным перцептроном	1,02		0,03	0,09	0,9	
Тема 14.3. Простая сверточная нейронная сеть для MNIST	1,02		0,03	0,09	0,9	
Тема 14.4. Большая сверточная нейронная сеть для MNIST	1,02		0,03	0,09	0,9	
Раздел 15. ПРЕДСТАВЛЕНИЕ СЛОВ В ВЕКТОРНОМ ПРОСТРАНСТВЕ	**3,06**		**0,09**	**0,27**	**2,7**	ПК-П1.1 ПК-П1.2 ПК-П1.3
Тема 15.1. Векторизация слов	1,02		0,03	0,09	0,9	ПК-П12.1 ПК-П12.2
Тема 15.2. Embedding слой Keras	1,02		0,03	0,09	0,9	ПК-П12.3 ПК-П15.1
Тема 15.3. Пример обучения векторизации	1,02		0,03	0,09	0,9	ПК-П15.2 ПК-П15.3

Раздел 16. LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ	**7,14**		**0,21**	**0,63**	**6,3**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 16.1. Рекуррентные нейронные сети	1,02		0,03	0,09	0,9	
Тема 16.2. Полностью рекуррентная сеть	1,02		0,03	0,09	0,9	
Тема 16.3. Проблема долгосрочных зависимостей	1,02		0,03	0,09	0,9	
Тема 16.4. LSTM сети	1,02		0,03	0,09	0,9	
Тема 16.5. Главная идея LSTM	1,02		0,03	0,09	0,9	
Тема 16.6. Разновидности LSTM сетей	1,02		0,03	0,09	0,9	
Тема 16.7. Прогнозирование временных рядов	1,02		0,03	0,09	0,9	
Раздел 17. КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ	**2,04**		**0,06**	**0,18**	**1,8**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 17.1. Проблема классификации последовательностей, ее сложность и решаемая задача	1,02		0,03	0,09	0,9	
Тема 17.2. Описание набора данных и решения задачи	1,02		0,03	0,09	0,9	
Раздел 18. НЕЙРОННЫЕ СЕТИ НА ОСНОВЕ БИБЛИОТЕКИ TENSORFLOW	**8,16**		**0,24**	**0,72**	**7,2**	ПК-П1.1 ПК-П1.2 ПК-П1.3 ПК-П12.1 ПК-П12.2 ПК-П12.3 ПК-П15.1 ПК-П15.2 ПК-П15.3
Тема 18.1. Начало работы с TensorFlow	1,02		0,03	0,09	0,9	
Тема 18.2. Основы работы в TensorFlow	1,02		0,03	0,09	0,9	
Тема 18.3. Определение вычислительных графов в TensorFlow	1,02		0,03	0,09	0,9	
Тема 18.4. Визуализация вычислительного графа с помощью TensorBoard	1,02		0,03	0,09	0,9	
Тема 18.5. Математика с TensorFlow	1,02		0,03	0,09	0,9	
Тема 18.6. Тензорные операции	1,02		0,03	0,09	0,9	
Тема 18.7. Матричные операции	1,02		0,03	0,09	0,9	
Тема 18.8. Пример нейронной сети в TensorFlow	1,02		0,03	0,09	0,9	
Итого	**68**	**1**	**2**	**6**	**59**	

5. Содержание разделов, тем дисциплин

Раздел 1. НЕЙРОНЫ И ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ
(Заочная: Внеаудиторная контактная работа - 1ч.; Лекционные занятия - 0,04ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 2,3ч.; Очная: Внеаудиторная контактная работа - 1ч.; Лекционные занятия - 0,4ч.; Практические занятия - 0,4ч.; Самостоятельная работа - 0,6ч.)

Тема 1.1. Комбинирование входных сигналов
(Заочная: Внеаудиторная контактная работа - 1ч.; Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 1,2ч.; Очная: Внеаудиторная контактная работа - 1ч.; Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
1. Комбинирование входных сигналов

Тема 1.2. Функция активации элемента.
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 1,1ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Функция активации элемента.

Раздел 2. ИСТОРИЯ НЕЙРОННЫХ СЕТЕЙ
(Заочная: Лекционные занятия - 0,04ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 1,8ч.; Очная: Лекционные занятия - 0,4ч.; Практические занятия - 0,4ч.; Самостоятельная работа - 0,6ч.)

Тема 2.1. Работы Дональда Олдинга Хебба, Уоррена Мак-Каллока и Уолтера Питтса
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Большое влияние на развитие искусственных нейронных сетей повлиял труд "Организация поведения" канадского ученого-психолога Дональда Олдинга Хебба (годы жизни 1904 - 1985). Один из основных постулатов данной работы - это ".. основой обучения нейронной сети мозга является синаптическая пластичность - способность клеток мозга увеличивать или уменьшать связи между собой под воздействием тех или иных факторов посредством увеличения или уменьшения связи в синапсе..". Хебб предположил, что любые два нейрона или система нейронов увеличивают связь между собой, если они одновременно возбуждены, и уменьшают связь, если один нейрон возбуждён, а другой нет. Когда одна клетка неоднократно вызывает возбуждение второй, то аксон первой клетки увеличивает выброс нейромедиатора в синапс либо увеличивает его площадь. (где-то взял, непомню где...).

Но первой ключевой работой по искусственным нейроным сетям стал труд 1943 года "Логическое исчисление идей, относящихся к нервной активности" американских ученых Уоррена Мак-Каллока и Уолтера Питтса. Ими был предложен формальный персептрон.

Тема 2.2. Работы Френка Розенблатта
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Другим пионером нейросетевой науки был американский нейрофизиолог Френк Розенблатт. который предложил и, главное, реализовал однослойную нейронную сеть из формальных персептронов (немного видоизменных по сравнению с МакКаллоком и Питтсом). Под руководством Розенблатта была программно реализована (на компьютере Марк-1 в Корнелльском университете) нейронная сеть распознающая геометрические фигуры (круг и квадрат)

Раздел 3. КЛАССИФИКАЦИЯ НЕЙРОННЫХ СЕТЕЙ
(Заочная: Лекционные занятия - 0,06ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 2,7ч.; Очная: Лекционные занятия - 0,6ч.; Практические занятия - 0,6ч.; Самостоятельная работа - 0,9ч.)

Тема 3.1. Классификация нейронных сетей по характеру обучения
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Классификация нейронных сетей по характеру обучения делит их на:
- нейронные сети, использующие обучение с учителем;
- нейронные сети, использующие обучение без учителя.

Тема 3.2. Классификация нейронных сетей по типу настройки весов
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Классификация нейронных сетей по типу настройки весов делит их на:
- сети с фиксированными связями – весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи;
- сети с динамическими связями – для них в процессе обучения происходит настройка синаптических весов.

Тема 3.3. Классификация нейронных сетей по типу входной информации
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
Классификация нейронных сетей по типу входной информации делит их на:
- аналоговые – входная информация представлена в форме действительных чисел;
- двоичные – вся входная информация в таких сетях представляется в виде нулей и единиц.

Раздел 4. АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ
(Заочная: Лекционные занятия - 0,04ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 1,8ч.; Очная: Лекционные занятия - 0,4ч.; Практические занятия - 0,4ч.; Самостоятельная работа - 0,6ч.)

Тема 4.1. Полносвязные нейронные сети
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
В полносвязных нейронных сетях каждый нейрон передает свой выходной сигнал остальным нейронам, в том числе и самому себе. Все входные сигналы подаются всем нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети.

Тема 4.2. Многослойные нейронные сети
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)

В многослойных (слоистых) нейронных сетях нейроны объединяются в слои. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в слое может быть любым и не зависит от количества нейронов в других слоях. В общем случае сеть состоит из слоев, пронумерованных слева направо. Внешние входные сигналы подаются на входы нейронов входного
слоя (его часто нумеруют как нулевой), а выходами сети являются выходные

сигналы последнего слоя. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько скрытых слоев. Связи от выходов нейронов некоторого слоя q к входам нейронов следующего слоя (q+1) называются последовательными.

Раздел 5. ФОРМАЛЬНЫЙ НЕЙРОН
(Заочная: Лекционные занятия - 0,06ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 2,7ч.; Очная: Лекционные занятия - 0,6ч.; Практические занятия - 0,6ч.; Самостоятельная работа - 1,1ч.)

Тема 5.1. Параметры нейрона
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,3ч.)
У нейрона есть несколько входных каналов и только один выходной канал. По входным каналам на нейрон поступают данные задачи, а на выходе формируется результат работы. Нейрон вычисляет взвешенную сумму входных сигналов, а затем преобразует полученную сумму с помощью заданной нелинейной функции. Множество, состоящее из порогового уровня и всех весов, называют параметрами нейрона.

Тема 5.2. Функция активации нейрона
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,4ч.)
Функция активации нейрона - это функция, которая вычисляет выходной сигнал нейрона. На вход этой функции подается сумма всех произведений сигналов и весов этих сигналов.

Наиболее часто используемые функции активации.
а) Пороговая функция. Это простая кусочно-линейная функция. Если входное значение меньше порогового, то значение функции активации равно минимальному допустимому, иначе – максимально допустимому.
б) Линейный порог. Это несложная кусочно-линейная функция. Имеет два линейных участка, где функция активации тождественно равна минимально допустимому и максимально допустимому значению и есть участок, на котором функция строго монотонно возрастает.
в) Сигмоидальная функция или сигмоида (sigmoid). Это монотонно возрастающая дифференцируемая S-образная нелинейная функция. Сигмоида позволяет усиливать слабые сигналы и не насыщаться от сильных сигналов.
г) Гиперболический тангенс (hyperbolic tangent, tanh). Эта функция принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от –1 до 1. Подобно сигмоиде,
гиперболический тангенс может насыщаться. Однако, в отличие от сигмоиды, выход данной функции центрирован относительно нуля.

Тема 5.3. Недостатки формального нейрона
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,4ч.)

• Предполагается, что нейрон мгновенно вычисляет свой выход, поэтому с помощью таких нейронов нельзя моделировать непосредственно системы с внутренним состоянием.
• Формальные нейроны, в отличие от биологических, не могут обрабатывать информацию синхронно.
• Нет четких алгоритмов выбора функции активации.
• Невозможно регулировать работу всей сети.
• Излишняя формализация понятий «порог» и «весовые коэффициенты». У реальных нейронов порог меняется динамически, в зависимости от активности нейрона и общего состояния сети, а весовые коэффициенты изменяются в зависимости от проходящих сигналов.

Раздел 6. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

(Заочная: Лекционные занятия - 0,04ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 1,8ч.; Очная: Лекционные занятия - 0,4ч.; Практические занятия - 0,4ч.; Самостоятельная работа - 0,8ч.)

Тема 6.1. Сеть из одного нейрона

(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,4ч.)

Один нейрон может выполнять простейшие вычисления, но основные функции нейросети обеспечиваются не отдельными нейронами, а соединениями между ними.

Тема 6.2. Однослойный перцептрон Френка Розенблатта

(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,4ч.)

Однослойный перцептрон представляет собой простейшую сеть, которая состоит из группы нейронов, образующих слой. Входные данные кодируются вектором значений, каждый элемент подается на соответствующий вход каждого нейрона в слое. В свою очередь, нейроны вычисляют выход независимо друг от друга. Размерность выхода (то есть количество элементов) равна количеству нейронов, а количество синапсов у всех нейронов должно быть одинаково и совпадать с размерностью входного сигнала.

Раздел 7. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

(Заочная: Лекционные занятия - 0,06ч.; Практические занятия - 0,28ч.; Самостоятельная работа - 2,7ч.; Очная: Лекционные занятия - 0,8ч.; Практические занятия - 0,8ч.; Самостоятельная работа - 1,6ч.)

Тема 7.1. Что такое обучение нейронной сети

(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,1ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,2ч.; Практические занятия - 0,2ч.; Самостоятельная работа - 0,4ч.)

Обучение нейронной сети - это процесс, в котором параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки параметров. Различают алгоритмы обучения с учителем и без учителя. Процесс обучения с учителем представляет собой предъявление сети выборки обучающих примеров. Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети.

Тема 7.2. Алгоритмы обучения нейронной сети с учителем и без учителя
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Для того, чтобы нейронная сети была способна выполнить поставленную задачу, ее необходимо обучить. Различают алгоритмы обучения с учителем и без учителя. Процесс обучения с учителем представляет собой предъявление сети выборки обучающих примеров. Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети. Затем по определенному правилу вычисляется ошибка, и происходит изменение весовых коэффициентов связей внутри сети в зависимости от выбранного алгоритма. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

Тема 7.3. Итерационные алгоритмы обучения нейронных сетей
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Для решения этой задачи могут использоваться следующие (итерационные) алгоритмы:

1. алгоритмы локальной оптимизации с вычислением частных производных первого порядка:
- градиентный алгоритм (метод наискорейшего спуска),
- методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента,
- метод сопряженных градиентов,
- методы, учитывающие направление антиградиента на нескольких шагах алгоритма;

2. алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка:
- метод Ньютона,
- методы оптимизации с разреженными матрицами Гессе,
- квазиньютоновские методы,
- метод Гаусса-Ньютона,
- метод Левенберга-Марквардта и др.;

3. стохастические алгоритмы оптимизации:
- поиск в случайном направлении,
- имитация отжига,
- метод Монте-Карло (численный метод статистических испытаний);

4. алгоритмы глобальной оптимизации (задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция).

Раздел 8. МНОГОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ
(Заочная: Лекционные занятия - 0,04ч.; Практические занятия - 0,18ч.; Самостоятельная работа - 1,8ч.; Очная: Лекционные занятия - 0,6ч.; Практические занятия - 0,6ч.; Самостоятельная работа - 1,2ч.)

Тема 8.1. Алгоритм обратного распространения ошибки
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Алгоритм обратного распространения ошибки является одним из методов обучения многослойных нейронных сетей прямого распространения. Обучение алгоритмом обратного распространения ошибки предполагает два прохода по всем слоям сети: прямого и обратного. При прямом проходе входной вектор подается на входной слой нейронной сети, после чего распространяется по сети от слоя к слою. В результате генерируется набор выходных сигналов, который и является фактической реакцией сети на данный входной образ. Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода все синаптические веса настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в направлении, обратном направлению синаптических связей. Отсюда и название – алгоритм обратного распространения ошибки. Синаптические веса настраиваются с целью максимального приближения выходного сигнала сети к желаемому.

Тема 8.2. Алгоритм обучения многослойной НС
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

1. Задаются шаг обучения α (0< α <1) и желаемая среднеквадратичная ошибка сети Em.
2. Инициализируются случайным образом весовые коэффициенты wi,jk и пороговые bjk значения НС.
3. Подаются последовательно образы из обучающей выборки на вход нейронной сети. При этом для каждого образа выполняются следующие действия:

a. Производится фаза прямого распространения входного образа по нейронной сети. Вычисляется выходное значение всех нейронов Yjk.
b. Вычисляются ошибки ϒj нейронов выходного и скрытого слоев.
c. Производится изменение весовых коэффициентов и порогов нейронных элементов для каждого слоя нейронной сети.

4. Вычисляется суммарная ошибка нейронной сети E
5. Если E>Em, то происходит переход к шагу 3, иначе выполнение алгоритма завершается

Раздел 9. ВВЕДЕНИЕ В KERAS И ЕГО ОСНОВНЫЕ ПРИНЦИПЫ
(Заочная: Лекционные занятия - 0,15ч.; Практические занятия - 0,54ч.; Самостоятельная работа - 5,4ч.; Очная: Лекционные занятия - 1,8ч.; Практические занятия - 1,8ч.; Самостоятельная работа - 3,6ч.)

Тема 9.1. Что такое Keras и глубинное обучение?
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Keras является высокоуровневыми нейронными сетями API, написанный на Python и могут работать поверх TensorFlow, CNTK или Теано. Он был разработан с упором на возможность быстрого экспериментирования. Способность идти от идеи к результату с наименьшей возможной задержкой является ключом к проведению хороших исследований.

Грубо говоря, глубинное обучение – просто более удобное название для искусственных нейросетей. «Глубинное» в этом словосочетании обозначает степень сложности (глубины) нейросети, которая зачастую может быть весьма поверхностной.
Создатели первой нейросети вдохновлялись структурой коры головного мозга. Базовый уровень сети, перцептрон, является по сути математическим аналогом биологического нейрона. И, как и в головном мозге, в нейросети могут появляться пересечённые друг с другом перцептроны.
Первый слой нейросети называется входным. Каждый узел этого слоя получает на вход какую-либо информацию и передает ее на последующие узлы в других слоях. Чаще всего между узлами одного слоя нет связей, а последний узел цепочки выводит результат работы нейросети.
Узлы посередине называются скрытыми, поскольку не имеют соединений с внешним миром, как узлы вывода и ввода. Они вызываются только в случае активации предыдущих слоев.

Глубинное обучение – это по сути техника обучения нейросети, которая использует множество слоев для решения сложных проблем (например, распознавания речи) с помощью шаблонов. В восьмидесятых годах большинство нейросетей были однослойными в силу высокой стоимости и ограниченности возможностей данных.
Если рассматривать машинное обучение как ответвление или вариант работы искусственного интеллекта, то глубинное обучение – это специализированный тип такого ответвления.
Машинное обучение использует компьютерный интеллект, который не дает ответа сразу. Вместо этого код будет запускаться на тестовых данных и, исходя из правильности их результатов, корректировать свой ход. Для успешности этого процесса обычно используются разнообразные техники, специальное программное обеспечение и информатика, описывающая статические методы и линейную алгебру.

Тема 9.2. Методы глубинного обучения
(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)
Методы глубинного обучения делятся на два основных типа:
- Обучение с учителем
- Обучение без учителя

Первый способ использует специально отобранные данные, чтобы добиться желаемого результата. Он требует довольно много человеческого вмешательства, ведь данные приходится выбирать вручную. Однако он удобен для классификации и регрессии.
Представьте, что вы владелец компании и хотите определить влияние премий на продолжительность контрактов с вашими подчиненными. При наличии заранее собранных данных, метод обучения с учителем был бы незаменим и очень эффективен.
Второй же способ не подразумевает заранее заготовленных ответов и алгоритмов работы. Он направлен на выявление в данных скрытых шаблонов. Обычно его используют для кластеризации и ассоциативных задач, например для группировки клиентов по поведению. «С этим также выбирают» на Amazon
– вариант ассоциативный задачи.
В то время как метод обучения с учителем довольно часто вполне удобен, его более сложный вариант все же лучше. Глубинное обучение зарекомендовало себя как нейросеть, не нуждающаяся в надзоре человека.

Тема 9.3. Важность глубинного обучения

(Заочная: Лекционные занятия - 0,02ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Компьютеры уже давно используют технологии распознавания определенных черт на изображении. Однако результаты были далеки от успеха.

Компьютерное зрение оказало на глубинное обучение невероятное влияние. Именно эти две техники в данный момент решают все задачи на распознавание. В частности, в распознавании лиц на фотографиях с помощью глубинного обучения преуспел Facebook. Это не простое улучшение технологии, а поворотный момент, изменяющий все более ранние представления: «Человек может с вероятностью в 97.53% определить, один ли человек представлен на двух разных фотографиях. Программа, разработанная командой Facebook, может делать это с вероятностью в 97.25% вне зависимости от освещения или
того, смотрит ли человек прямо в камеру или повернут к ней боком».

Распознавание речи тоже претерпело значительные изменения. Команда Baidu – одного из лидирующих поисковиков Китая – разработала систему распознавания речи, сумевшую опередить человека в скорости и точности написания текста на мобильных устройствах. На английском и мандаринском.

Что особенно занимательно – написание общей нейросети для двух абсолютно разных языков не потребовало особенного труда: «Так исторически сложилось, что люди видели Китайский и Английский, как два совершенно разных языка, поэтому и подход к каждому из них требовался различный», — говорит начальник исследовательского центра Baidu, Andrew Ng. «Алгоритмы обучения сейчас настолько обобщены, что вы можете просто обучаться».

Google использует глубинное обучение для управления энергией в дата- центрах компании. Они смогли сократить затраты ресурсов для охлаждения на 40%. Это около 15% повышения эффективности энергопотребления и миллионы долларов экономии.

Тема 9.4. Микросервисы глубинного изучения

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Вот краткий обзор сервисов, связанных с глубинным обучением.

Illustration Tagger. Дополненный Illustration2Vec, этот сервис позволяет отмечать изображения с рейтингом «защищенный», «сомнительный»,
«опасный», «копирайт» или «общий» для того, чтобы заранее понять содержание картинки.

Классификатор возраста использует технологии анализа фотографии для определения возраста человека. Places 365 Classifier использует заранее натренированную нейросеть в сочетании с базой данных за 2016 год для определения местоположение человека по фотографии (например, деревня, аптека, номер гостиной, горы и так далее). Не стоит забывать и о InceptionNet – прямом наследнике InceptionNet от Google. Эта нейросеть на основе анализа фотографии машины выдает пять лучших моделей, соответствующих этому автомобилю.

Тема 9.5. Open Source фреймворки о глубинном обучении

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Доступность глубинного обучения обеспечена несколькими проектами с открытым исходным кодом. В этом списке есть как известные технологии, так и менее популярные. Он составлялся на основе направленности нейросети, сложности и академичности. Вот этот список:
DeepLearning4j(DL4J):
- Основана на JVM
- Свободное распространение
- Интегрируется с Hadoop и Spark Theano:
- Популярна на Academia
- Сказочно простая

- Редактируется на Python и Numpy Torch:
- Основана на Lua
- Домашняя версия используется компаниями Facebook и Twitter
- Содержит заранее натренированные модели TensowFlow:
- Дополнение для Theano от Google
- Редактируется на Python и Numpy
- Зачастую применяется для решения определенного спектра проблем Caffe:
- Не общего назначения. Основной упор на машинное зрение
- Редактируется на C++
- Есть интерфейс на Python

Здесь мы подробно рассматриваем Keras. Используйте Keras, если вам нужна библиотека глубокого обучения, которая:
1. Позволяет легко и быстро создавать прототипы (благодаря удобству, модульности и расширяемости).
2. Поддерживает как сверточные сети, так и повторяющиеся сети, а также комбинации этих двух.
3. Легко работает на процессоре и графическом процессоре.

Тема 9.6. Основные принципы Keras
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Установка и настройка. Изначально Keras вырос как удобная надстройка над Theano. Отсюда и его греческое имя — κέρας, что значит "рог" по-гречески, что, в свою очередь, является отсылкой к Одиссее Гомера. Хотя, с тех пор утекло много воды, и Keras стал сначала поддерживать Tensorflow, а потом и вовсе стал его частью.
Keras устанавливается как обычный питоновский пакет: pip install keras
ВНИМАНИЕ: Чтобы работать с Keras, у вас уже должен быть установлен хотя бы один из фреймворков — Theano или Tensorflow.

Бэкенды — это то, из-за чего Keras стал известен и популярен. Фронтенд (англ. front-end) — клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса. Бекенд (англ. back-end) — программно- аппаратная часть сервиса. Фронт- и бекенд — это вариант архитектуры программного обеспечения. Термины появились в программной инженерии вследствие развития принципа разделения ответственности между внешним представлением и внутренней реализацией. Back-end создает некоторое API, которое использует front-end. Таким образом front-end разработчику не нужно знать особенностей реализации сервера, а back-end разработчику — реализацию front-end. Keras позволяет использовать в качестве бэкенда разные другие фреймворки. При этом написанный код будет исполняться независимо от используемого бэкенда. Начиналась разработка, как уже было сказано, с Theano, но со временем добавился Tensorflow. Сейчас Keras по умолчанию работает именно с ним, но если нужно использовать Theano, то есть два варианта, как это сделать:
1. Отредактировать файл конфигурации keras.json, который лежит по пути $HOME/.keras/keras.json(или %USERPROFILE%\.keras\keras.json в случае операционных систем семейства Windows). Нам нужно поле backend:
{
"image_data_format": "channels_last", "epsilon": 1e-07,
"floatx": "float32", "backend": "theano"
}
2. Второй путь — это задать переменную окружения KERAS_BACKEND, например, так:
KERAS_BACKEND=theano python -c "from keras import backend" Using Theano backend.

Удобство для пользователя. Keras - это API, предназначенный для людей, а не для машин. Он ставит пользовательский интерфейс спереди и в центре. Keras следует наилучшим методам снижения когнитивной нагрузки: он предлагает последовательные и простые API, он минимизирует количество действий пользователя, необходимых для случаев общего использования, и обеспечивает четкую и эффективную обратную связь с ошибкой пользователя.
Модульность. Под моделью понимается последовательность или график автономных полностью настраиваемых модулей, которые могут быть подключены вместе с минимальными ограничениями. В частности, нейронные слои, функции затрат, оптимизаторы, схемы инициализации, функции активации, схемы регуляризации - это автономные модули, которые вы можете комбинировать для создания новых моделей.
Легкая масштабируемость. Новые модули просто добавлять (как новые классы и функции), а существующие модули предоставляют множество примеров. Чтобы иметь возможность легко создавать новые модули, вы можете полностью выразить свою выразительность, что делает Keras подходящим для передовых исследований.
Работа с Python. Нет отдельных файлов конфигурации моделей в декларативном формате. Модели описаны в коде Python, который компактен, легче отлаживается и обеспечивает простоту расширяемости.

Раздел 10. МОДЕЛИ KERAS
(Заочная: Лекционные занятия - 0,12ч.; Практические занятия - 0,36ч.; Самостоятельная работа - 3,6ч.; Очная: Лекционные занятия - 1,2ч.; Практические занятия - 1,2ч.; Самостоятельная работа - 2,4ч.)

Тема 10.1. Основная структура данных Keras

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Основная структура данных Keras - это модель, способ организации слоев. В Keras доступны два основных типа моделей: последовательная модель Sequential и класс Model, используемый с функциональным API. Простейшим типом модели является Sequential модель, которая представляет собой линейную совокупность слоев. Для более сложных архитектур необходимо использовать функциональный API Keras, который позволяет создавать произвольные графики слоев.

Тема 10.2. Краткая характеристка моделей Keras

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Эти модели имеют ряд общих свойств и общих методов:

- model.layers - представляет собой список слоев, содержащихся в модели.
- model.inputs - представляет собой список входных тензоров модели.
- model.outputs - это список выходных тензоров модели.
- model.summary() - печатает сводное представление о модели.
- model.get_config() - возвращает словарь, содержащий конфигурацию модели.
- model.get_weights() - возвращает список всех весовых тензоров в модели.
- model.set_weights(weights) - устанавливает значения весов модели, из массива. Массивы в списке должны иметь ту же форму, что и возвращаемые get_weights().
- model.to_json() - возвращает представление модели как в виде строки JSON. Это представление не включает веса, а только архитектуру.

Приведем пример использования метода model.to_json():

```
from keras.models import model_from_json json_string = model.to_json()
model = model.from_json(json_string)
```

Тема 10.3. API класса Model

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Используя функциональный API можно создать экземпляр класса Model для некоторого входного тензора и выходной тензора используя следующий код:

```
from keras.models import Model
from keras.layers import Input, Dense a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

Эта модель будет включать все уровни, необходимые для вычисления b
на основе a.

В случае моделей с несколькими входами или с несколькими выходами также можно использовать списки:

```
model = Model(inputs=[a1, a2], outputs=[b1, b2, b3])
```

Тема 10.4. Основные методы класса Model

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Рассмотрим наиболее важные методы класса Model, необходимые для организации процесса обучения нейронных сетей.

1. Метод настройки модели для обучения:

compile(self, optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=None, weighted_metrics=None, target_tensors=None)

2. Обучение модели для определенного количества эпох:

fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=None)

Основные аргументы этого метода:

- x: массив данных обучения (если модель имеет один вход) или список массивов (если модель имеет несколько входов).
- y: массив целевых данных (если модель имеет один вывод) или список массивов (если модель имеет несколько выходов).
- batch_size: количество выборок на обновление градиента. Если не указано, batch_size будет по умолчанию установлено значение 32.
- epochs: Количество эпох для обучения модели.
- validation_split: Float между 0 и 1. Доля данных обучения, которые будут использоваться в качестве данных валидации. Модель будет выделять эту часть данных обучения, не будет тренироваться на ней и будет оценивать ошибку и любые модельные показатели по этим данным в конце каждой эпохи.
- initial_epoch: эпоха, с которой начать обучение (полезно для возобновления предыдущего цикла обучения).

3. Метод для оценки качества обученности модели. Этот метод возвращает значения ошибок и показателей для модели в тестовом режиме.

evaluate(self, x=None, y=None, batch_size=None, verbose=1, sample_weight=None, steps=None)

4. Метод для создания выходных прогнозов для входных выборок.

predict(self, x, batch_size=None, verbose=0, steps=None) Основные аргументы:

- x: входные данные, как в виде массива (или список массивов Numpy, если модель имеет несколько входов).
- steps: общее количество шагов (партии выборок) до объявления раунда прогнозирования.

5. Метод извлечения слоя на основе его имени или индекса. Этот метод возвращает экземпляр слоя.

get_layer(self, name=None, index=None) Основные аргументы:

- name: String, имя слоя.
- index: Integer, индекс слоя.

Раздел 11. СЛОИ В KERAS

(Заочная: Лекционные занятия - 0,39ч.; Практические занятия - 0,36ч.; Самостоятельная работа - 3,6ч.; Очная: Лекционные занятия - 1,2ч.; Практические занятия - 1,2ч.; Самостоятельная работа - 2,4ч.)

Тема 11.1. Общие методы слоев

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Все слои Keras имеют ряд общих методов:

- layer.get_weights()- возвращает веса слоя в виде списка массивов Numpy.
- layer.set_weights(weights)- устанавливает веса слоя из списка массивов (с теми же формами, что и выход get_weights).
- layer.get_config() - возвращает словарь, содержащий конфигурацию слоя.

Слой может быть восстановлен из его конфигурации используя следующий:

```
layer = Dense(32)

config = layer.get_config()
reconstructed_layer = Dense.from_config(config)
```

Если слой имеет один узел (т. е. если он не является общим слоем), то можно получить его входной тензор, выходной тензор, размерность входного массива и размерность выходного массива через свойства:

- layer.input
- layer.output
- layer.input_shape
- layer.output_shape

Тема 11.2. Плотный слой Dense

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Слой Dense реализует операцию: output = activation(dot(input, kernel) + bias) где activation функция активации, переданная в качестве activation аргумента, kernel является матрицей слоя весов, и bias представляет собой вектор смещения, созданный слоем.
Плотный слой создается использованием метода:
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)
Рассмотрим пример создания плотного слоя. #сначала создаем последовательную модель
model = Sequential()
# добавляем первый плотный слой
# модель будет принимать на входе массив (*, 16) и выходной массив (*, 32)
model.add(Dense(32, input_shape=(16,)))
# при добавлении следующих слоев нет необходимости указывать размеры входных массивов
model.add(Dense(32))

Чтобы указать функцию активации, которая будет применена к выходу необходимо использовать метод:
keras.layers.Activation(activation)
В качестве аргументы activation необходимо указать имя используемой функции активации.
Переобучение (overfitting) — одна из проблем глубоких нейронных сетей, состоящая в том, что модель хорошо распознает только примеры из обучающей выборки, адаптируясь к обучающим примерам, вместо того чтобы учиться классифицировать примеры, не участвовавшие в обучении (теряя способность к обобщению). Наиболее эффективным решением проблемы переобучения является метод исключения (Dropout).
keras.layers.Dropout(rate, noise_shape=None, seed=None)
Сети для обучения получаются с помощью исключения из сети (dropping out) нейронов с вероятностью rate, таким образом, вероятность того, что нейрон останется в сети, составляет 1- rate. “Исключение” нейрона означает, что при любых входных данных или параметрах он возвращает 0.
Для преобразования результата в определенную форму необходимо использовать метод:
keras.layers.Reshape(target_shape)
В качестве аргумента target_shape указывается кортеж целых чисел. Рассмотрим пример:
model.add(Reshape((3, 4), input_shape=(12,)))
# размерность массива выходного слоя: model.output_shape == (None, 3, 4)
model.add(Reshape((6, 2)))
# размерность массива выходного слоя: model.output_shape == (None, 6, 2) Для изменения размеров входного массива можно использовать метод: keras.layers.Permute(dims)
Этот метод полезен, например, для соединения RNN и коннектов вместе.
Пример
model = Sequential()

model.add(Permute((2, 1), input_shape=(10, 64)))

Тема 11.3. Сверточные слои
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Слой свёртки — это основной блок свёрточной нейронной сети. Слой свёртки включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты матричного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения. Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении.

1. Conv1D - Этот слой создает сверточное ядро, по одному пространственному (или временному) измерению: keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', data_format='channels_last', dilation_rate=1, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

Основные аргументы:

- filters: размерность выходного пространства (т. е. количество выходных фильтров в свертке).
- kernel_size: целое или список целых чисел, определяющий длину окна свертки.
- strides: целое или список целых чисел, определяющий длину шага свертки.
- activation: функция активации слоя. Если этот параметр не указан, то активация не применяется (т.е. «линейная» функция активации a(x) = x).

2. Conv2D – это 2D сверточный слой (например, пространственная свертка над изображениями). Этот слой создает ядро свертки для создания тензора выходов.
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True,

kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)

3. Conv3D - 3D сверточный слой (например, пространственная свертка над объемами). Этот слой создает ядро свертки, которое свернуто со слоем ввода для создания тензора выходов.
keras.layers.Conv3D(filters, kernel_size, strides=(1, 1, 1), padding='valid', data_format=None, dilation_rate=(1, 1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None).
При создании этого слоя если use_bias= True, тогда вектор смещения создается и добавляется к выходу. При использовании этого слоя в качестве первого слоя в модели в качестве аргумента input_shape необходимо указать кортеж целых чисел, который не включает ось выборки, например, input_shape=(128, 128, 128, 1).

Тема 11.4. Слой пулинга

(Заочная: Лекционные занятия - 0,3ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Слой пулинга представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя, проходя нелинейное преобразование. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения. Пулинг интерпретируется так. Если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться. Слой пулинга, как правило, вставляется после слоя свёртки перед слоем следующей свёртки.
Наиболее употребительна при этом функция максимума.

1. MaxPooling1D
keras.layers.MaxPooling1D(pool_size=2, strides=None, padding='valid') Основные аргументы:
- pool_size: Integer, размер максимальных окон объединения.
- strides: параметр, с помощью которого можно уменьшить масштаб. Например, 2 уменьшит вдвое вход.

2. MaxPooling2D
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
Операция объединения для пространственных данных.
3. MaxPooling3D
keras.layers.MaxPooling3D(pool_size=(2, 2, 2), strides=None, padding='valid', data_format=None)
Операция объединения трехмерных данных (пространственное или пространственно-временное объединение).

Раздел 12. ОСНОВЫ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОЙ МОДЕЛЬЮ KERAS
(Заочная: Лекционные занятия - 0,15ч.; Практические занятия - 0,45ч.; Самостоятельная работа - 4,5ч.; Очная: Лекционные занятия - 1,5ч.; Практические занятия - 1,5ч.; Самостоятельная работа - 3ч.)

Тема 12.1. Создание последовательной модели (Sequential)
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Создать последовательную модель (Sequential модель) можно передав список экземпляров слоя в конструктор класса Sequential:
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential([Dense(32, input_shape=(784,)), Activation('relu'), Dense(10), Activation('softmax'),])
Также можно просто добавить слои с помощью метода add(). model = Sequential()
model.add(Dense(32, input_dim=784)) model.add(Activation('relu'))

Тема 12.2. Указание размерности входных данных
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Модель должна знать размерность входного массива данных. Поэтому первый слой в Sequential модели (и только первый, потому что следующие слои могут получать автоматически эту информацию из предыдущего слоя) должен получать информацию о размерность входного массива. Существует несколько способов сделать это:

- Передать аргумент input_shape первому слою.
- Некоторые 2D-слои, например, Dense, поддерживают спецификацию их формы ввода через аргумент input_dim, а некоторые 3D- слои поддерживают аргументы input_dim и input_length.

Таким образом, следующие фрагменты кода строго эквивалентны:
model.add(Dense(32, input_shape=(784,))) model.add(Dense(32, input_dim=784))

Тема 12.3. Компиляция

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Перед подготовкой модели необходимо настроить процесс обучения, который выполняется с помощью метода compile. Этом метод имеет три аргумента:

- Оптимизатор. Это может быть строковый идентификатор существующего оптимизатора (например, rmsprop или adagrad) или экземпляр класса Optimizer.
- Функция вычисления ошибок. Это цель, которую модель попытается свести к минимуму. Он может быть строковым идентификатором существующей функции ошибок (например, categorical_crossentropy или mse) или может быть целевой функцией.
- Список метрик. Для любой проблемы классификации вы захотите установить это metrics=['accuracy']. Метрика может быть строковым идентификатором существующей метрики или специальной метрической функцией.

Примеры:

1. Для задачи классификации по нескольким классам

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

2. Для задачи бинарной классификации model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])

3. Для регрессионный задачи со среднеквадратичной ошибкой model.compile(optimizer='rmsprop', loss='mse')

Пример создания и использования собственной метрики:
import keras.backend as K
def mean_pred(y_true, y_pred): return K.mean(y_pred)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy', mean_pred])

Тема 12.4. Обучение

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Модели Keras обучаются массивам входных данных и целевых значении.
Для этого необходимо использовать функцию fit.
Рассмотрим пример создания модели для бинарной классификации.
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100)) model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
# Создаем фиктивные данные
import numpy as np
data = np.random.random((1000, 100)) labels = np.random.randint(2, size=(1000, 1))
# Обучаем модель в 10 эпох по 32 примера в пакете
model.fit(data, labels, epochs=10, batch_size=32)

Тема 12.5. Пример многослойного перцептрона (MLP) для многоклассовой классификации
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation from keras.optimizers import SGD
import numpy as np
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)), num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=20)) model.add(Dropout(0.5))
model.add(Dense(64, activation='relu')) model.add(Dropout(0.5)) model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(x_train, y_train, epochs=20, batch_size=128) score = model.evaluate(x_test, y_test, batch_size=128)
```

Раздел 13. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В KERAS

(Заочная: Лекционные занятия - 0,09ч.; Практические занятия - 0,27ч.; Самостоятельная работа - 2,7ч.; Очная: Лекционные занятия - 0,9ч.; Практические занятия - 0,9ч.; Самостоятельная работа - 1,8ч.)

Тема 13.1. Сверточная нейронная сеть
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Сверточные нейронные сети являются одной из форм многослойных нейронных сетей. Здесь приведена схема типичного CNN. Первая часть состоит

из слоев свертки и максимального пула, которые выступают в качестве экстрактора признаков. Вторая часть состоит из полносвязного слоя, который выполняет нелинейные преобразования извлеченных признаков и действует как классификатор.

На приведенной выше диаграмме вход подается в сеть последовательных слоев Conv, Pool и Dense. Выходной сигнал может быть слоем softmax, указывающим, есть ли кошка или что-то еще. Также, а качестве выходного может быть использован сигмоидный слой, на выходе которого будет вероятность того, что изображение будет кошкой. Рассмотрим слои более подробно.
Сверточный слой можно рассматривать как глаза сверточной нейронной сети. Нейроны в этом слое ищут определенные особенности. Свертку можно рассматривать как взвешенную сумму между двумя сигналами или функциями. Пример операции свертки на матрице размером 5 × 5 с ядром размером 3 × 3 показан ниже. Ядро свертки скользит по всей матрице для

получения карты активации.

Предположим, что входное изображение имеет размер 32x32x3, т.е. это трехмерный массив глубины 3. Любой фильтр свертки, который мы определяем на этом слое, должен иметь глубину, равную глубине ввода. Поэтому мы можем выбрать фильтры свертки глубины 3 (например, 3x3x3 или 5x5x3 или 7x7x3 и т. Д.). Выберем фильтр свертки размера 3x3x3, т.е сверточное ядро будет кубом вместо квадрата.
Если мы сможем выполнить операцию свертки, сдвинув фильтр 3x3x3 на все изображение размером 32x32x3, то мы получим изображение с разрешением 30x30x1. Это связано с тем, что операция свертки невозможна для полосы шириной 2 пикселя вокруг изображения. Фильтр всегда находится внутри изображения и поэтому 1 пиксель удаляется от левой, правой, верхней и нижней части изображения.
Для входного изображения 32x32x3 и размера фильтра 3x3x3 у нас есть 30x30x1 местоположения, и для каждого местоположения существует нейрон. Тогда выходы 30x30x1 или активации всех нейронов называются

картами активации. Карта активации одного уровня служит входом для следующего слоя.
В нашем примере есть 30 × 30 = 900 нейронов, потому что есть много мест, где может применяться фильтр 3x3x3. В отличие от традиционных нейронных сетей, где веса и смещения нейронов независимы друг от друга, в случае сверточных нейронных сетей, нейроны, соответствующие одному фильтру в слое, имеют одинаковые веса и смещения. В приведенном выше случае мы сдвигаем окно на 1 пиксель за раз. Мы также можем сдвинуть окно более чем на 1 пиксель. Это число называется шагом.
Как правило, используют более одного фильтра в одном слое свертки. Если мы используем 32 фильтра, у нас будет карта активации размером 30x30x32.
Обратите внимание, что все нейроны, связанные с одним и тем же фильтром, имеют одинаковые веса и смещения. Таким образом, количество весов при использовании 32 фильтров - это 3x3x3x32 = 288, а число смещений - 32.
На картинке показаны 32 карты активации, полученные от применения

сверточных ядер.
Как вы можете видеть, после каждой свертки результат уменьшается по размеру (так как в этом случае мы переходим от 32 × 32 до 30 × 30). Для удобства стандартная практика заключается в том, чтобы накладывать нули на границу входного слоя таким образом, чтобы выход был такого же размера, как и входной. Итак, в этом примере, если мы добавим дополнение размером 1 по

обе стороны от входного слоя, размер выходного уровня будет 32x32x32, что упростит реализацию.

Тема 13.2. Набор данных - CIFAR10
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Набор данных CIFAR10 поставляется вместе с Keras. Он имеет 50 000 учебных образов и 10000 тестовых изображений 10 классов, таких как самолеты, автомобили, птицы, кошки, олени, собаки, лягушки, лошади, корабли и грузовики. С помощью следующего программного кода можно осуществить загрузку и подготовку данных CIFAR10 для дальнейшей обработки с помощью нейронных сетей:

```
from  future  import print_function import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten from keras.layers import Conv2D, MaxPooling2D
import os batch_size = 32
num_classes = 10
epochs = 100
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models') model_name = 'keras_cifar10_trained_model.h5'
# Разделяем данные на обучающий и тестовый наборы: (x_train, y_train), (x_test, y_test) = cifar10.load_data() print('x_train shape:', x_train.shape) print(x_train.shape[0], 'train samples') print(x_test.shape[0], 'test samples')
# Преобразование векторов классов в двоичные матрицы y_train = keras.utils.to_categorical(y_train, num_classes) y_test = keras.utils.to_categorical(y_test, num_classes)
```

Изображения имеют размер 32 × 32. На рисунке приведены несколько примеров.

Сверточная нейронная сеть будет состоять из сверточных слоев, и слоев MaxPooling. Мы также включим Dropout слой для избежания переобучения. На выходе сети мы добавим полносвязный слой (Dense), за которым следует слой softmax. Здесь приведен программный код создания структуры модели. В приведенном выше коде мы используем 6 сверточных слоев и 1 полносвязный слой. Сначала в модель добавляем сверточные слои с 32 фильтрами с размером окна 3 × 3. Далее мы добавляем сверточный слой с 64 фильтрами. За каждым слоем добавлен слой максимального пуллинга с размером окна 2 × 2. Также добавлены слои Dropout с коэффициентами 0,25 и
0.5 для того чтобы не произошло переобучение сети. В заключительных строках мы добавляем плотный слой Dense, который выполняет классификацию среди 10 классов с использованием функции активации softmax.

```
model = Sequential()
model.add(Conv2D(32,(3,3),padding='same', input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3))) model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25)) model.add(Conv2D(64, (3, 3), padding='same')) model.add(Activation('relu')) model.add(Conv2D(64, (3, 3))) model.add(Activation('relu')) model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.25))
model.add(Flatten()) model.add(Dense(512)) model.add(Activation('relu')) model.add(Dropout(0.5)) model.add(Dense(num_classes)) model.add(Activation('softmax')) model.summary()
```

Если мы выведем информацию о структуре модели, мы увидим следующую таблицу с подробным описанием каждого слоя.

```
Layer (type)    Output Shape    Param #
=================================================================
====
```

Тема 13.3. Обучение сети

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Поскольку это проблема классификации по 10 классам, мы будем использовать категорическую потерю энтропии и использовать оптимизатор RMSProp для обучения сети.

```
# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6) # Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy']) x_train = x_train.astype('float32')
x_test = x_test.astype('float32') x_train /= 255
x_test /= 255
//Запустим его на количество эпох epochs.
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test), shuffle=True)

# Сохраняем модель и веса
if not os.path.isdir(save_dir): os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name) model.save(model_path)
print('Saved trained model at %s ' % model_path)


# Проверяем точность работы модели.
scores = model.evaluate(x_test, y_test, verbose=1) print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Раздел 14. РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

(Заочная: Лекционные занятия - 0,12ч.; Практические занятия - 0,36ч.; Самостоятельная работа - 3,6ч.; Очная: Лекционные занятия - 1,2ч.; Практические занятия - 1,2ч.; Самостоятельная работа - 2,4ч.)

Тема 14.1. Описание набора данных (датасет) MNIST

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

MNIST - это набор данных, разработанный Янном ЛеКуном, Коринной Кортес и Кристофер Бургес для оценки моделей машинного обучения по проблеме классификации рукописных цифр. Набор данных был построен из ряда отсканированных наборов документов, доступных в Национальном институте стандартов и технологий (NIST). Изображения цифр были взяты из множества отсканированных документов, нормированных по размеру и по центру. Это делает его отличным набором данных для оценки моделей, позволяя разработчику сосредоточиться на механизме обучения с очень небольшой очисткой данных или необходимой подготовкой.

Каждое изображение представляет собой квадрат размером 28 на 28 пикселей (всего 784 пикселя). В этом наборе 60 000 изображений используются для обучения модели, и для ее тестирования используется отдельный набор из

10 000 изображений. Это задача распознавания 10 цифр (от 0 до 9) или классификация на 10 классов.

Библиотека глубокого обучения Keras предоставляет удобный метод mnist.load_data() для загрузки набора данных MNIST. Набор данных загружается автоматически при первом вызове этой функции и сохраняется в вашем домашнем каталоге в ~ / .keras /datasets /mnist.pkl.gz в виде файла 15 МБ. Это очень удобно для разработки и тестирования моделей глубокого обучения.

Чтобы продемонстрировать, насколько легко загружать набор данных MNIST, мы сначала напишем небольшой скрипт для загрузки и визуализации первых четырех изображений в наборе учебных материалов.

```
from keras.datasets import mnist import matplotlib.pyplot as plt
# load (downloaded if needed) the MNIST dataset (X_train, y_train), (X_test, y_test) = mnist.load_data() # plot 4 images as gray scale
plt.subplot(221)

plt.imshow(X_train[0], cmap=plt.get_cmap('gray')) plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray')) plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray')) plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray')) # show the plot
plt.show()
```

Запустив приведенный выше пример, вы должны увидеть изображение ниже.

Тема 14.2. Базовая модель с многослойным перцептроном

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Чтобы понять действительно ли нам нужна сложная модель, такая как сверточная нейронная сеть сначала попробуем использовать очень простую модель нейронной сети с одним скрытым слоем. Мы будем использовать эту

сеть как основу для сравнения более сложных сверточных моделей нейронных сетей.
Начнем с импорта классов и функций, которые нам понадобятся.
import numpy
from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.utils import np_utils
Учебный набор данных структурирован как трехмерный массив. Чтобы подготовить данные, сперва мы представим изображения в виде одномерных массивов (так как считаем каждый пиксель отдельным входным признаком). В этом случае изображения размером 28 × 28 будут преобразованы в массивы, содержащие 784 элементов.
Мы можем сделать это преобразование, используя функцию reshape() библиотеки NumPy. Для уменьшения потребления оперативной памяти преобразуем точность значений пикселей в 32.
(X_train, y_train), (X_test, y_test) = mnist.load_data() num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32') X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
Значения пикселей заданы в оттенках серого со значениями от 0 до 255. Для эффективного обучения нейронных сетей практически всегда рекомендуется выполнять некоторое масштабирование входных значений. Мы можем нормализовать значения пикселей в диапазоне 0 и 1, разделив каждое значение на максимальные значения 255.
X_train = X_train / 255 X_test = X_test / 255
Выходная переменная представляет собой целое число от 0 до 9, т.к. это задача классификации с несколькими классами. Хорошей практикой является

использование кодирования значений класса преобразованием вектора целых чисел класса в двоичную матрицу.
Мы можем легко сделать это, используя встроенную вспомогательную функцию np_utils.to_categorical () в Keras.
y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]
Теперь создадим нашу простую модель однослойной нейронной сети и определим ее в функции.
def baseline_model(): # create model
model = Sequential() model.add(Dense(num_pixels,input_dim=num_pixels, kernel_initializer='normal', activation='relu'))
model.add(Dense(num_classes,kernel_initializer='normal', activation='softmax')) model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy']) return model
Модель представляет собой простую нейронную сеть с одним скрытым слоем с таким же количеством нейронов, что и количество входов (784). В скрытом слое используем полулинейную функцию активации relu.
На выходном слое используется функция активации softmax для преобразования выходов в вероятностные значения и позволяет выбрать один класс из 10 в качестве выходного значения модели. Теперь нам осталось только определить функцию потерь, алгоритм оптимизации и метрики, которые мы будет собирать. В задачах с вероятностной классификацией, в качестве функции потерь лучше всего использовать не квадратичную ошибку, а перекрестную энтропию. Потери будут меньше для вероятностных задач (например, с логистической/softmax

функцией для выходного слоя), в основном из-за того, что данная функция предназначена для максимизации уверенности модели в правильном определении класса, и ее не заботит распределение вероятностей попадания образца в другие классы. Используемый алгоритм оптимизации будет напоминать какую-то форму алгоритма градиентного спуска, отличие

Тема 14.3. Простая сверточная нейронная сеть для MNIST
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Мы узнали, как загрузить набор данных MNIST и как запрограммировать простую многослойную модель персептрона, и теперь настало время разработать более сложную сверточную нейронную сеть.
В этом разделе мы создадим простую CNN для MNIST, которая продемонстрирует, как использовать все аспекты современной реализации CNN.
Первый шаг - импортировать необходимые классы и функции.
import numpy
from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D from keras.utils import np_utils
from keras import backend as K K.set_image_dim_ordering('th')
Далее инициализируем генератор случайных чисел на постоянное начальное значение для воспроизводимости результатов.

seed = 7 numpy.random.seed(seed)
Затем нам нужно загрузить набор данных MNIST и изменить его, чтобы он был подходящим для обучения CNN.
(X_train, y_train), (X_test, y_test) = mnist.load_data() # reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32') Как и прежде нормализуем значения пикселей в диапазоне 0 и 1. X_train = X_train / 255
X_test = X_test / 255
y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]
Затем мы определяем модель нейронной сети. Сверточные нейронные сети более сложны, чем стандартные многослойные персептроны, поэтому мы начнем с использования простой структуры
Ниже представлена архитектура сети.
1. Первый скрытый слой - это сверточный слой, Convolution2D. Этот слой имеет 32 карты функций, размер которых равен 5 × 5 и функции активации relu.
2. Затем мы определяем слой пулинга maxPooling2D с размером пула 2 × 2, который дает максимальные значения.
3. Следующий уровень - это уровень регуляризации Dropout. Он настроен на случайное исключение 20% нейронов в слое, чтобы уменьшить переобучение.
4. Далее - слой, который преобразует данные двумерной матрицы в вектор, называемый Flatten. Он позволяет обрабатывать выходные данные стандартными полносвязными слоями.
5. Затем полносвязный слой с 128 нейронами и функцией активации relu.

6. Наконец, выходной слой имеет 10 нейронов для 10 классов и функцию активации softmax для вывода вероятностных результатов распознавания для каждого класса.
def baseline_model(): # create model
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.2)) model.add(Flatten())
model.add(Dense(128, activation='relu')) model.add(Dense(num_classes, activation='softmax')) # Compile model
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['ac curacy'])
return model
Как и в примере с моногослойным персептроном эта модель 10 эпох обучения, при каждом обновлении весов используется 200 изображений.
model = baseline_model()
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbose=2)
scores = model.evaluate(X_test, y_test, verbose=0) print("CNN Error: %.2f%%" % (100-scores[1]*100))

Тема 14.4. Большая сверточная нейронная сеть для MNIST
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Теперь, когда мы увидели, как создать простую сверточную нейронную сеть, давайте создадим модель, которая может быть близка к новейшим научным результатам.

В начале программы импортируем классы и функции, затем загружаем и готовим данные так же, как в предыдущем примере CNN.

```
import numpy
from keras.datasets import mnist from keras.models import Sequential from keras.layers import Dense from keras.layers import Dropout from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D from keras.utils import np_utils
from keras import backend as K K.set_image_dim_ordering('th') seed = 7

numpy.random.seed(seed) # load data
(X_train, y_train), (X_test, y_test) = mnist.load_data() # reshape to be [samples][pixels][width][height]
X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32') # normalize inputs from 0-255 to 0-1
X_train = X_train / 255 X_test = X_test / 255
y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) num_classes = y_test.shape[1]
```

На этот раз мы создадим большую архитектуру сверточной нейронной сети с дополнительными сверточными слоями, слоями пуллинга и полностью связанными слоями. Сетевую топологию можно резюмировать следующим образом.

1. Сверточный слой Conv2D с 30 функциональными картами размером 5 × 5.
2. Слой максимального пулинга MaxPooling2D размером 2 * 2.
3. Сверточный слой Conv2D с 15 картинными картами размером 3 × 3.
4. Слой максимального пулинга MaxPooling2D размером 2 * 2.
5. Слой исключения Dropout с вероятностью 20%.
6. Слой Flatten.
7. Полносвязный слой Dense с 128 нейронами и функцией активации relu.
8. Полносвязный слой Dense с 50 нейронами и функцией активации relu.
9. Выходной полносвязный слой Dense с функцией активации softmax.

```
def larger_model():
# create model model = Sequential()
model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(15, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Dropout(0.2)) model.add(Flatten())
model.add(Dense(128, activation='relu')) model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['ac curacy'])
return model
model = larger_model()
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200)
scores = model.evaluate(X_test, y_test, verbose=0) print("Large CNN Error: %.2f%%" % (100-scores[1]*100))
```

Эта модель уже достигает уровня ошибки классификации 0,89%.

Раздел 15. ПРЕДСТАВЛЕНИЕ СЛОВ В ВЕКТОРНОМ ПРОСТРАНСТВЕ
(Заочная: Лекционные занятия - 0,09ч.; Практические занятия - 0,27ч.; Самостоятельная работа - 2,7ч.; Очная: Лекционные занятия - 0,9ч.; Практические занятия - 0,9ч.; Самостоятельная работа - 1,8ч.)

Тема 15.1. Векторизация слов
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)
Векторизация слов (world embeding) - это класс подходов для представления слов и документов с использованием векторного представления. Это улучшение по сравнению с традиционными схемами кодирования, где для представления каждого слова использовались большие разреженные векторы или оценка каждого слова в векторе для представления целого словарного запаса. Эти представления были скудными, потому что словари были обширными, и данное слово или документ представлялось бы большим вектором, состоящим в основном из нулевых значений.
Вместо этого в world embeding слова представлены плотными векторами, где вектор представляет проекцию слова в непрерывное векторное пространство.

Представление слова в векторном пространстве получается из текста и основывается на словах, которые окружают слово, когда оно используется.
Два популярных примера методов вложения слов в текст включают:
- Word2Vec.
- GloVe.

В дополнение к этим ранее разработанным методам, векторизацию слов можно изучить как часть модели глубокого обучения.

Тема 15.2. Embedding слой Keras
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Keras предлагает слой Embedding, который можно использовать в моделях нейронных сетей для обработки текстовых данных. Он требует, чтобы входные данные были закодированы целыми числами, так что каждое слово представлено уникальным целым числом. Эта стадия подготовки данных может быть выполнена с использованием API Tokenizer, также предоставляемого Keras.
Слой Embedding инициализируется случайными весами и производит векторизацию для всех слов в наборе учебных данных.
Это гибкий слой, который можно использовать различными способами, такими как:
1. Его можно использовать отдельно, чтобы изучить векторизацию слов, которое может быть сохранено и использовано в другой модели позже.
2. Он может использоваться как часть модели глубокого обучения, в которой векторизацию изучается вместе с самой моделью.
3. Его можно использовать для загрузки предварительно подготовленной модели векторизации слов, типа передачи обучения.
Слой векторизации Embedding определяется как первый скрытый уровень сети. Он имеет три аргумента:
• input_dim: Это размер словаря текстовых данных. Например, если целые данные кодируются значениями от 0 до 10, то размер словаря будет составлять 11 слов.

• output_dim: Это размерность векторного пространства, в которое будут векторизироваться слова. Он определяет размер выходных векторов этого слоя для каждого слова. Например, это может быть 32 или 100 или даже больше.
• input_length: Это длина входных последовательностей, как вы бы определили для любого входного слоя модели Keras. Например, если все входные документы состоят из 1000 слов, это будет 1000.
Например, ниже мы определяем слой Embedding со словарем в 200 слов (например, целочисленные кодированные слова от 0 до 199 включительно), векторное пространство из 32 измерений, в которое будут векторизованы слова, и входные документы, каждая из которых содержит 50 слов.
e = Embedding(200, 32, input_length=50)
В слое Embedding содержатся веса, которые можно впоследствии можно анализировать. Если вы сохраните модель в файле, это будет включать в себя весы для слоя Embedding.
Выходом слоя Embedding является 2D-вектор с одним вектором для каждого слова во входной последовательности слов (входной документ).
Если необходимо подключить полносвязный слой непосредственно к слою Embedding, то вы должны сначала сгладить матрицу 2D-вывода на 1D- вектор, используя слой Flatten.
Теперь давайте посмотрим, как мы можем использовать слой Embedding
на практике.

Тема 15.3. Пример обучения векторизации
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Мы создадим небольшую задачу, в которой у нас есть 10 текстовых документов, каждый из которых имеет комментарий о части работы, выполненной студентом. Каждый текстовый документ классифицируется как положительный «1» или отрицательный «0». Сначала мы определим документы и их метки классов.

```
from numpy import array

from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences from keras.models import Sequential
from keras.layers import Dense from keras.layers import Flatten
from keras.layers.embeddings import Embedding
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!', 'Weak', 'Poor effort!', 'not good', 'poor work', 'Could have done better.']
labels = array([1,1,1,1,1,0,0,0,0,0])
```

Затем мы можем закодировать каждый документ целыми числами. Это означает, что в качестве входных данных слой Embedding будет иметь последовательности целых чисел. Keras предоставляет функцию one_hot(), которая создает хэш каждого слова как эффективное целочисленное кодирование. Мы оценим размер словаря 50, что намного больше, чем необходимо для снижения вероятности столкновений совпадений от хэш-функции.

```
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs] print(encoded_docs)
```

Последовательности имеют разную длину, и поэтому мы будем заполнять все входные последовательности до длины 4. Это мы можем сделать это со встроенной функцией Keras pad_sequences().

```
max_length = 4
padded_docs=pad_sequences(encoded_docs,maxlen=max_length, padding='post')
print(padded_docs)
```

Теперь мы готовы определить наш слой Embedding как часть нашей модели нейронной сети. Модель представляет собой простую модель двоичной классификации. Важно отметить, что выход из слоя Embedding будет 4 вектора по 8 измерений каждый, по одному для каждого слова.

```
model = Sequential()

model.add(Embedding(vocab_size, 8, input_length=max_length)) model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(model.summary())
```

Наконец, мы можем подгонять и оценивать классификационную модель.

```
model.fit(padded_docs, labels, epochs=150, verbose=0)
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0) print('Accuracy: %f' % (accuracy*100))
```

Запуск примера сначала печатает целочисленное кодирование документов. [[42, 38], [24, 27], [26, 11], [3, 27], [36], [42], [28, 11], [8, 24], [28, 27],
[23, 4, 38, 31]]

Затем печатаются заполненные вектора каждого документа, заполненные нулями чтобы они были одинаковой длины.

```
[[42 38 0  0]
[24 27  0  0]
[26 11  0  0]
[ 3 27  0  0]
[36 0  0  0]
[42 0  0  0]
[28 11  0  0]
[ 8 24 0 0]
[28 27 0 0]
[23 4 38 31]]
```

После определения сети будет напечатана информация о структуре сети. Как и ожидалось,

Раздел 16. LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ
(Заочная: Лекционные занятия - 0,21ч.; Практические занятия - 0,63ч.; Самостоятельная работа - 6,3ч.; Очная: Лекционные занятия - 2,1ч.; Практические занятия - 2,1ч.; Самостоятельная работа - 4,2ч.)

Тема 16.1. Рекуррентные нейронные сети
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)
Рекуррентные нейронные сети (англ. Recurrent neural network; RNN) — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки.

В отличие от многослойных перцептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на сегменты, например, распознавание рукописного текста или распознавание речи. Было предложено много различных архитектурных решений для рекуррентных сетей от простых до сложных. В последнее время наибольшее распространение получили сеть с долговременной и кратковременной памятью (LSTM) и управляемый рекуррентный блок (GRU).

В диаграмме выше участок нейронной сети A получает некие данные X на вход и подает на выход некоторое значение H. Циклическая связь позволяет передавать информацию от текущего шага сети к следующему.
Существует много разновидностей, решений и конструктивных элементов рекуррентных нейронных сетей. Трудность рекуррентной сети заключается в том, что если учитывать каждый шаг времени, то становится необходимым для каждого шага времени создавать свой слой нейронов, что вызывает серьёзные вычислительные сложности. Кроме того, многослойные реализации оказываются вычислительно неустойчивыми, так как в них как правило исчезают или зашкаливают веса. Если ограничить расчёт фиксированным временным окном, то полученные модели не будут отражать долгосрочных трендов. Различные подходы пытаются усовершенствовать модель исторической памяти и механизм запоминания и забывания.
Рекуррентные нейронные сети не так уж сильно отличаются от обычных нейронных сетей. Их можно представить себе, как множество копий одной и

той же сети, причем, каждая копия передает сообщение следующей копии. Посмотрите, что получится, если мы развернем цикл:

Такая “цепная” сущность показывает, что рекуррентные нейронные сети по природе своей тесно связаны с последовательностями и списками.

Тема 16.2. Полностью рекуррентная сеть
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Это базовая архитектура разработана в 1980-х. Сеть строится из узлов, каждый из которых соединён со всеми другими узлами. У каждого нейрона порог активации меняется со временем и является вещественным числом. Каждое соединение имеет переменный вещественный вес. Узлы разделяются на входные, выходные и скрытые.
Для обучения с учителем с дискретным временем, каждый (дискретный) шаг времени на входные узлы подаются данные, а прочие узлы завершают свою активацию, и выходные сигналы готовятся для передачи нейроном следующего уровня. Если, например, сеть отвечает за распознавание речи, в результате на выходные узлы поступают уже метки (распознанные слова).
В обучении с подкреплением (reinforcement learning) нет учителя, обеспечивающего целевые сигналы для сети, вместо этого иногда используется функция годности[en] или функция оценки (reward function), по которой проводится оценка качества работы сети, при этом значения на выходе оказывает влияние на поведение сети на входе. В частности, если сеть реализует игру, на выходе измеряется количество пунктов выигрыша или оценки позиции. Каждая цепочка вычисляет ошибку как суммарную девиацию по выходным сигналам сети. Если имеется набор образцов обучения, ошибка вычисляется с учётом ошибок каждого отдельного образца.

Тема 16.3. Проблема долгосрочных зависимостей
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)
Одна из идей, которая делает РНС столь притягательными, состоит в том, что они могли бы использовать полученную в прошлом информацию для текущих задач. Например, они могли бы использовать предыдущие кадры видео для понимания последующих. Иногда нам достаточно недавней информации, чтобы выполнять текущую задачу. Например, представим модель языка, которая пытается предсказать следующее слово, основываясь на предыдущих. Если мы пытаемся предсказать последнее слово в предложении “Тучи на небе”, нам не нужен больше никакой контекст - достаточно очевидно, что в конце предложения речь идёт о небе. В таких случаях, где невелик промежуток между необходимой информацией и местом, где она нужна, РНС могут научиться использовать информацию, полученную ранее.

Но также бывают случаи, когда нам нужен более широкий контекст. Предположим, нужно предсказать последнее слово в тексте “Я вырос во Франции… Я свободно говорю по французски”. Недавняя информация подсказывает, что следующее слово, вероятно, название языка, но если мы хотим уточнить, какого именно, нам нужен предыдущий контекст вплоть до информации о Франции. Совсем не редко промежуток между необходимой информацией и местом, где она нужна, становится очень большим. К сожалению, по мере роста промежутка, РНС становятся неспособны научиться соединять информацию.

Теоретически, РНС способны обрабатывать такие долговременные зависимости. Человек может тщательно подобрать их параметры, чтобы решать игрушечные проблемы такой формы. Однако, на практике, РНС не способны выучить такое.

Тема 16.4. LSTM сети
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Сети долго-краткосрочной памяти (Long Short Term Memory) - обычно просто называют “LSTM” - особый вид РНС, способных к обучению долгосрочным зависимостям. Они работают невероятно хорошо на большом разнообразии проблем и в данный момент широко применяются. LSTM специально спроектированы таким образом, чтобы избежать проблемы долгосрочных зависимостей. Запоминать информацию на длительный период времени - это практически их поведение по-умолчанию, а не что-то такое, что они только пытаются сделать. В LSTM сетях удалось обойти проблему исчезновения или зашкаливания градиентов в процессе обучения методом обратного распространения ошибки. Сеть LSTM обычно управляется с помощью рекуррентных вентилей, которые называются вентили (gates) «забывания». Ошибки распространяются назад по времени через потенциально неограниченное количество виртуальных слоёв. Таким образом происходит обучение в LSTM, при этом сохраняя память о тысячах и даже миллионах временных интервалов в прошлом. Топологии сетей типа LSTM могут разрабатываться в соответствии со спецификой задачи. В сети LSTM даже большие задержки между значимыми событиями могут

учитываться, и тем самым высокочастотные и низкочастотные компоненты могут смешиваться.
Все рекуррентные нейронные сети имеют форму цепи повторяющих модулей (repeating module) нейронной сети. В стандартной РНС эти повторяющие модули будут иметь очень простую структуру, например, всего один слой гиперболического тангенса (tanhtanh).

LSTM тоже имеют такую цепную структуру, но повторяющий модуль имеет другое строение. Вместо одного нейронного слоя их четыре, причем они взаимодействуют особым образом.

Здесь введены следующие обозначения:

В диаграмме выше каждая линия передает целый вектор от выхода одного узла к входам других. Розовые круги представляют поточечные операторы, такие как сложение векторов, в то время, как желтые

прямоугольники - это обученные слои нейронной сети. Сливающиеся линии обозначают конкатенацию, в то время как ветвящиеся линии обозначают, что их содержимое копируется, и копии отправляются в разные места.

Тема 16.5. Главная идея LSTM
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Ключ к LSTM - клеточное состояние (cell state) - горизонтальная линия, проходящая сквозь верхнюю часть диаграммы. Клеточное состояние - это что- то типа ленты конвейера. Она движется прямо вдоль всей цепи только лишь с небольшими линейными взаимодействиями. Информация может просто течь по ней без изменений.

LSTM имеет способность удалять или добавлять информацию к клеточному состоянию, однако эта способность тщательно регулируется структурами, называемыми вентилями (gates). Вентили - это способ избирательно пропускать информацию. Они составлены из сигмоидного слоя НС и операции поточечного умножения (pointwise multiplication).

Сигмоидный слой подает на выход числа между нулем и единицей, описывая таким образом, насколько каждый компонент должен быть пропущен сквозь вентиль. Ноль - “ничего не пропускать”, один - “пропускать все”. LSTM имеет три таких вентиля, чтобы защищать и контролировать клеточное состояние.
Первым шагом в нашей LSTM будет решить какую информацию мы собираемся выбросить из клеточного состояния. Это решение принимается сигмоидным слоем, называемым “забывающим вентилем” (“forget gate layer”). Он получает на входе значения ht−1и xt и подает на выход число между 0 и 1. Единица означает “сохрани это полностью”, в тот время как ноль означает “избавься от этого полностью”.
Давайте вернемся к нашему примеру языковой модели, пытающейся предсказать следующее слово, основываясь на всех предыдущих. В такой проблеме клеточное состояние может включать род подлежащего, что позволит использовать правильные формы местоимений. Когда мы видим новое подлежащее, мы забываем род предыдущего подлежащего.
Следующим шагом будет решить, какую новую информацию мы собираемся сохранить в клеточном состоянии. Этот шаг состоит из двух частей. Во-первых, сигмоидный слой, называемый “входным вентилем” (“input gate layer”), решает, какие значения мы обновим. Далее, слой гирпеболического тангенса создает вектор кандидатов на новые значения σt, который может быть добавлен к состоянию. На следующем шаге мы соединим эти две части, чтобы создать обновление для состояния.

В примере с нашей языковой моделью мы бы хотели добавить род нового подлежащего к клеточному состоянию, чтобы заменить род старого, которое мы должны забыть.
Теперь пришла пора обновить старое клеточное состояние, σt-1 новым клеточным состоянием σt. Все решения уже приняты на предыдущих шагах, осталось только сделать это. Мы умножаем старое состояние на ft, забывая все, что мы ранее решили забыть. В случае с языковой моделью, это как раз то место, где мы теряем информацию о роде старого подлежащего и добавляем новую информацию, как решили на предыдущих шагах.

Наконец, нам нужно решить, какой результат мы собираемся подать на выход. Этот результат будет основан на нашем клеточном состоянии, но будет его отфильтрованной версией. Сначала мы запускаем сигмоидный слой, который решает, какие части клеточного состояния мы собираемся отправить на выход. Затем мы пропускаем клеточное состояние сквозь гиперболический тангенс (tanhtanh) (чтобы уместить значения в промежуток от −1до 1) и умножаем его на выход сигмоидного вентиля, так что мы отправляем на выход
только те части, которые мы хотим.

В примере с языковой моделью, если она только что видела подлежащее, она могла бы подать на выход информацию, относящуюся к глаголу (в случае, если следующее слово именно глагол). К примеру, она, возможно, подаст на выход число подлежащего (единственное или множественное). Таким образом, мы будем знать, какая форма глагола должна быть подставлена (если конечно дальше идет именно глагол).

Тема 16.6. Разновидности LSTM сетей
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

В одном из популярных вариантов LSTM добавляются “глазковые соединения” (“peephole connections”). Это значит, что мы позволяем вентилям “подглядывать” за клеточным состоянием.

Другая вариация - использование спаренных забывающих и входных вентилей. Вместо того, чтобы независимо решать, что забыть и куда мы должны добавить новую информацию, мы принимаем эти решения одновременно. Мы забываем что-то только в том случае, когда мы получаем что-то другое на это место. Мы получаем на вход новые значения только когда забываем что-то старое.

Несколько более существенно отличается от LSTM вентильная рекуррентная единица (Gated Recurrent Unit) или GRU. Она совмещает забывающие и входные вентили в один “обновляющий вентиль” (“update gate”). Она также сливает клеточное состояние со скрытым слоем и вносит некоторые другие изменения. Модель, получающаяся в результате, проще, чем обычная модель LSTM и она набирает популярность.

Это только некоторые из наиболее заметных вариантов LSTM. Есть множество других, например, глубинно-вентильные РНС (Depth Gated RNNs). Существует и совершенно другой подход к изучению долговременных зависимостей, например, часовые РНС (Clockwork RNNs)

Тема 16.7. Прогнозирование временных рядов
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Задачи прогнозирования временных рядов - сложный тип проблемы прогнозирующего моделирования. В отличие от регрессионного предсказательного моделирования временные ряды также добавляют сложность зависимости последовательности от входных переменных.
Мощный тип нейронной сети, предназначенный для обработки последовательностей называется рекуррентными нейронными сетями. Сеть с

длинной короткой памятью или сеть LSTM - это тип рекуррентной нейронной сети, используемой в глубоком обучении, потому что можно успешно обучать очень большие архитектуры.
В этом разделе мы разработаем ряд LSTM для стандартной задачи прогнозирования временных рядов. Эти примеры помогут вам разработать свои собственные структурированные LSTM-сети для задач прогнозирования временных рядов.
Задача, которую мы рассмотрим это - проблема прогнозирования пассажирских авиаперевозок. Задача состоит в том зная год и месяц предсказать количество пассажиров международных авиакомпаний.
Набор данных доступен бесплатно можно скачать с адреса https://datamarket.com/data/set/22u3/international-airline-passengers-monthly-totals-in-thousands-jan-49-dec-60#!ds=22u3&display=line с именем файла « international-airlines-passengers.csv ». Данные варьируются от января 1949 года до декабря 1960 года или 12 лет с 144 наблюдениями. Мы можем загрузить этот набор данных с помощью библиотеки Pandas. Нам не интересна дата, учитывая, что каждое наблюдение разделяется одним и тем же интервалом в один месяц. Поэтому, когда мы загружаем набор данных, мы можем исключить первый столбец. После загрузки мы можем легко построить весь набор данных. Код для загрузки и построения набора данных приведен ниже.

```
import pandas
import matplotlib.pyplot as plt
dataset = pandas.read_csv('international-airline-passengers.csv', usecols=[1], engine='python', skipfooter=3)
plt.plot(dataset) plt.show()
```

С течением времени можно увидеть восходящий тренд в наборе данных и некоторую периодичность для набора данных, который, вероятно, соответствует периоду отпуска в северном полушарии.

Мы можем сформулировать эту задачу как задачу регрессии. То есть, учитывая количество пассажиров (в тысячах единиц) в этом месяце прогнозировать количество пассажиров в следующем месяце. Мы можем написать простую функцию, чтобы преобразовать наш единственный столбец данных в двухстолбцовый набор данных: первая колонка, содержащая количество пассажиров и второй столбец, который будет содержать количество пассажиров в следующем месяце.
Прежде чем мы начнем, давайте сначала импортируем все функции и классы, которые мы намерены использовать.

```
import numpy
import matplotlib.pyplot as plt from pandas import read_csv import math
from keras.models import Sequential from keras.layers import Dense

from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean_squared_error
```

Прежде чем мы что-либо сделать инициализируем генератор случайных чисел, чтобы гарантировать, что наши результаты будут воспроизводимыми.

```
numpy.random.seed(7)
```

Используем код из предыдущего раздела для загрузки набора данных в виде данных Pandas. Затем мы можем извлечь массив NumPy из фрейма данных и преобразовать целочисленные значения в значения с плавающей запятой, которые более подходят для работы с нейронной сетью.

```
dataframe = read_csv('international-airline-passengers.csv',usecols=[1], engine='python', skipfooter=3)
```

Раздел 17. КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ

(Заочная: Лекционные занятия - 0,06ч.; Практические занятия - 0,18ч.; Самостоятельная работа - 1,8ч.; Очная: Лекционные занятия - 0,6ч.; Практические занятия - 0,6ч.; Самостоятельная работа - 1,2ч.)

Тема 17.1. Проблема классификации последовательностей, ее сложность и решаемая задача
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Классификация последовательностей - это проблема прогнозирующего моделирования, в которой у вас есть некоторая последовательность входов по пространству или времени, и задача заключается в прогнозировании категории для последовательности.

Сложность этой проблемы заключается в том, что последовательности могут варьироваться по длине, состоять из очень большого словарного запаса входных символов и могут потребовать от модели изучения долгосрочного контекста или зависимостей между символами во входной последовательности. В этом разделе мы разработаем LSTM рекуррентную модели нейронной сети для задач классификации последовательностей.

Задача, которую мы будем решать - это задача классификации тональности отзыва фильма IMDB. Каждый отзыв представляет собой переменную последовательность слов, и тональность каждого отзыва фильма должна быть классифицирована.

Тема 17.2. Описание набора данных и решения задачи
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Набор данных содержит 25 000 высокополярных отзывов фильмов (хорошие или плохие) для обучения и того же количества для тестирования. Keras содержит функция imdb.load_data (), которая позволяет загружать набор данных в формате, который готов для использования в нейронной сети. Слова были заменены целыми числами, которые указывают упорядоченную частоту каждого слова в наборе данных. Поэтому предложения в каждом обзоре состоят из последовательности целых чисел.
Будем отображать каждое слово на 32-значный вещественный вектор. Мы также ограничим общее количество слов, которые нас интересуют в моделировании, до 5000 наиболее часто встречающихся слов. И так как длина последовательности (количество слов) в каждом обзоре меняется мы будем

ограничивать каждый обзор 500 словами, усекая длинные обзоры и заполняя более короткие обзоры нулевыми значениями.
Начнем как обычно с импорта классов и функций, необходимых для этой модели, и инициализации генератора случайных чисел, чтобы мы могли легко воспроизвести результаты.
import numpy
from keras.datasets import imdb from keras.models import Sequential from keras.layers import Dense from keras.layers import LSTM
from keras.layers.embeddings import Embedding from keras.preprocessing import sequence numpy.random.seed(7)
Далее нам нужно загрузить набор данных IMDB. Мы ограничиваем набор данных до 5000 слов. Мы также разделили набор данных на поезд (50%) и тест (50%).
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
Затем нам нужно усечь или дополнить входные последовательности так, чтобы они были одинаковой длины для обучения нейронной сети. max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length) X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
Теперь мы можем создать, скомпилировать и обучить нашу модель LSTM.
Первый слой - это Embedded слой, который использует 32 вектора для представления каждого слова. Следующий слой - это слой LSTM содержащий
100 нейронов. Наконец, поскольку это проблема классификации, мы используем полносвязный Dense выходной слой с одним нейроном и функцией активации сигмоида, чтобы получить на выходе 0 или 1 для предсказания двух классов (хороших и плохих).

Поскольку это проблема двоичной классификации, используется логарифмическая потеря как функция ошибок (binary_crossentropy). Используется эффективный алгоритм оптимизации ADAM. Модель подходит обучение за 3 эпохи, потому что она быстро переобучается.
embedding_vecor_length = 32 model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length= max_review_length))
model.add(LSTM(100)) model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, epochs=3, batch_size=64)
После обучения сети мы оцениваем качество модели по тестовым данным.
scores = model.evaluate(X_test, y_test, verbose=0) print("Accuracy: %.2f%%" % (scores[1]*100))
Выполнение этого примера приводит к следующему результату.
Epoch 1/3 16750/16750 [==========] - 107s - loss: 0.5570 - acc: 0.7149 Epoch 2/3 16750/16750 [==========] - 107s - loss: 0.3530 - acc: 0.8577 Epoch 3/3 16750/16750 [==========] - 107s - loss: 0.2559 - acc: 0.9019
Accuracy: 86.79%

Раздел 18. НЕЙРОННЫЕ СЕТИ НА ОСНОВЕ БИБЛИОТЕКИ TENSORFLOW

(Заочная: Лекционные занятия - 0,24ч.; Практические занятия - 0,72ч.; Самостоятельная работа - 7,2ч.; Очная: Лекционные занятия - 2,4ч.; Практические занятия - 2,4ч.; Самостоятельная работа - 4,8ч.)

Тема 18.1. Начало работы с TensorFlow

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

TensorFlow - это библиотека программного обеспечения с открытым исходным кодом, созданная Google, которая используется для внедрения систем машинного обучения и глубокого обучения. Эти два имени содержат ряд мощных алгоритмов, которые разделяют общую задачу - позволить компьютеру узнать, как автоматически определять сложные шаблоны и / или

принимать наилучшие возможные решения. TensorFlow, в основе своей, является библиотекой для программирования потока данных. Он использует различные методы оптимизации, чтобы сделать вычисления математических выражений проще и эффективнее.

Некоторые из ключевых особенностей TensorFlow:

- Эффективно работает с математическими выражениями, включающими многомерные массивы
- Хорошая поддержка глубоких нейронных сетей и концепций машинного обучения
- Использование GPU / CPU, где один и тот же код может быть выполнен на обеих архитектурах
- Высокая масштабируемость вычислений на машинах и огромные массивы данных

Установить TensorFlow можно выполнив команду:

Загрузка и сборка TensorFlow может занять несколько минут.

Тема 18.2. Основы работы в TensorFlow

(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

В TensorFlow вычисление описывается с использованием графов потоков данных. Каждый узел графа представляет собой экземпляр математической операции (например, сложение, деление или умножение), и каждое ребро представляет собой многомерный набор данных (тензор), на котором выполняются операции.

Прежде чем перейти к обсуждению элементов TensorFlow, мы сначала создадим сеанс работы с TensorFlow, чтобы понять, как выглядит программа TensorFlow.
В TensorFlow константы создаются с использованием функции: constant(value, dtype=None, shape=None, name='Const', verify_shape=False), где value постоянное значение, которое будет использоваться при дальнейших вычислениях, dtype является параметром, указывающим тип данных (например, float32/64, int8/16), shape является необязательным параметром, указывающим размер массива данных, name является необязательным задающим имя для тензора. Если вам нужны константы с определенными значениями внутри вашей обучающей модели, тогда объект типа constant может использоваться как в следующем примере:
z = tf.constant(5.2, name="x", dtype=tf.float32)
Переменные в TensorFlow являются буферами в памяти, содержащими тензоры, которые должны быть явно инициализированы. Просто вызывая конструктор, мы добавляем переменную в вычислительный граф. Переменные

особенно полезны, как только вы начинаете с моделей обучения, и они используются для хранения и обновления параметров. Начальное значение, переданное в качестве аргумента конструктора, представляет собой тензор или объект, который может быть преобразован или возвращен как тензор. Это означает, что если мы хотим заполнить переменную некоторыми предопределенными или случайными значениями, которые будут использоваться впоследствии в процессе обучения и обновлены при итерациях, мы можем определить ее следующим образом:
k = tf.Variable(tf.zeros([1]), name="k")
Другой способ использования переменных в TensorFlow - это вычисления, когда эта переменная не является обучаемой и может быть определена следующим образом:
k = tf.Variable(tf.add(a, b), trainable= False)

Сеанс TensorFlow инкапсулирует управление и состояние среды выполнения TensorFlow. Сеанс без параметров будет использовать граф по умолчанию, созданный в текущем сеансе, иначе класс сеанса принимает параметр графа, который используется в этом сеансе для выполнения. Ниже приведен краткий фрагмент кода, в котором показано, как термины, определенные выше, могут использоваться в TensorFlow для вычисления простой линейной функции y=a*x+b:
import tensorflow as tf
x = tf.constant(-2.0, name="x", dtype=tf.float32) a = tf.constant(5.0, name="a", dtype=tf.float32) b = tf.constant(13.0, name="b", dtype=tf.float32) y = tf.Variable(tf.add(tf.multiply(a, x), b))
init = tf.global_variables_initializer() with tf.Session() as session:
session.run(init) print(session.run(y))

Тема 18.3. Определение вычислительных графов в TensorFlow
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Преимущество при работе с графами потоков данных заключается в том, что модель отделена от ее исполнения, т.е. неважно на каком вычислительном устройстве выполняется программный код, на процессоре, графическом процессоре или некоторой комбинации. Программа в среде TensorFlow может выполняться на процессоре или графическом процессоре, а вся сложность, связанная с кодом исполнения скрыта от пользователя. Граф вычисления - это встроенный процесс, который использует библиотеку, не требуя прямого вызова объекта графа. Граф вычислений может быть построен в процессе использования библиотеки TensorFlow без необходимости явно создавать объекты Graph.
Объект Graph в TensorFlow может быть создан в результате простой строки кода c = tf.add(a, b). Это код создаст операционный узел, который принимает два тензора a и b которые вычисляют их сумму c в качестве результата.
Плейсхолдер - это способ, позволяющий разработчикам вводить данные в график вычислений через заполнители, которые связаны внутри некоторых выражений. Сигнатура плейсхолдера:
placeholder(dtype, shape= None , name= None)
где dtype - тип элементов в тензорах.
Преимущество плейсхолдеров заключается в том, что они позволяют разработчикам создавать операции в вычислительном графе вообще, без необходимости заранее предоставлять данные, и данные могут быть добавлены во время выполнения из внешних источников.
Рассмотрим простую задачу умножения двух целых чисел x и y в TensorFlow и использованием плейсхолдера:
import tensorflow as tf
x = tf.placeholder(tf.float32, name="x") y = tf.placeholder(tf.float32, name="y")

z = tf.multiply(x, y, name="z") with tf.Session() as session:
print(session.run(z, feed_dict={x: 2.1, y: 3.0}))

Тема 18.4. Визуализация вычислительного графа с помощью TensorBoard
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

TensorBoard - это инструмент визуализации для анализа графиков потока данных. Он может быть полезно для лучшего понимания моделей машинного обучения. С TensorBoard вы можете получить представление о различных типах статистики о параметрах и подробностях о частях вычислительного графа. Глубокая нейронная сеть имеет большое количество узлов. TensorBoard позволяет разработчикам получить представление о каждом узле и о том, как вычисление выполняется во время выполнения TensorFlow.
Теперь давайте вернемся к нашему примеру где мы определили линейную функцию y = a*x + b.
Чтобы регистрировать события с сеанса, которые позже могут использоваться в TensorBoard, TensorFlow предоставляет класс FileWriter. Его можно использовать для создания файла событий для хранения сводок и событий. Конструктор FileWriter принимает шесть параметров и выглядит так:
init (logdir, graph=None, max_queue=10, flush_secs=120, graph_def=None, filename_suffix=None) - где требуется указать обязательный параметр logdir, а другие - значения по умолчанию. Параметр графа будет передан из объекта сеанса, созданного в программе. Полный код примера выглядит так:

```
import tensorflow as tf
x = tf.constant(-2.0, name="x", dtype=tf.float32) a = tf.constant(5.0, name="a", dtype=tf.float32) b = tf.constant(13.0, name="b", dtype=tf.float32) y = tf.Variable(tf.add(tf.multiply(a, x), b))
init = tf.global_variables_initializer()

with tf.Session() as session:
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("logs", session.graph) session.run(init)
print(session.run(y))
```

Мы добавили две новые строки в которых создается и используется объект FileWriter для вывода событий в файл, как описано выше.
После запуска программы у нас появляется файл в журналах каталогов, и для того чтобы посмотреть содержимое этого файла необходимо запустить tensorboard:

```
tensorboard --logdir logs/
```

После открытия http://localhost:6006 и нажатия на пункт меню «Графики»

(расположенный в верхней части страницы) вы сможете увидеть график, как на картинке ниже:

TensorBoard маркирует константы и сводные узлы конкретными символами, которые показаны ниже.
99

Тема 18.5. Математика с TensorFlow
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Тензоры - это основные структуры данных в TensorFlow, и они представляют собой соединительные грани в графе потока данных.
Тензор просто идентифицирует многомерный массив или список. Тензорную структуру можно идентифицировать тремя параметрами: рангом, размером и типом.

- Ранг: Определяет количество измерений тензора. Ранг известен как порядок или n-размерности тензора, где, например, тензор ранга 1 является тензором вектора или ранга 2, является матрицей.
- Размер: Размер тензора - это количество строк и столбцов.
- Тип: Тип данных элементов тензора.

Чтобы построить тензор в TensorFlow, мы можем построить n-мерный массив. Это можно сделать легко, используя библиотеку NumPy, или путем преобразования n-мерного массива Python в тензор TensorFlow.

Чтобы построить 1-мерный тензор, мы будем использовать массив
NumPy:

```
import numpy as np
tensor_1d = np.array([1.45, -1, 0.2, 102.1])
```

Работа с подобным массивом аналогична работе со встроенным списком Python. Основное отличие состоит в том, что массив NumPy также содержит некоторые дополнительные свойства, такие как размер, форма и тип.

```
>> print tensor_1d
[   1.45    -1. 0.2 102.1 ]
>> print tensor_1d[0] 1.45
>> print tensor_1d[2] 0.2
>> print tensor_1d.ndim 1
>> print tensor_1d.shape

(4,)
>> print tensor_1d.dtype float64
```

Массив NumPy можно легко преобразовать в тензор TensorFlow используя вспомогательную функцию convert_to_tensor, что помогает разработчикам преобразовывать объекты Python в объекты тензора. Эта функция принимает тензорные объекты, массивы NumPy и списки Python.

```
tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)
```

Теперь, если мы привяжем наш тензор к сеансу TensorFlow, мы сможем увидеть результаты нашего преобразования.

```
import numpy as np import tensorflow as tf
tensor_1d = np.array([1.45, -1, 0.2, 102.1])

tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)
```

```
with tf.Session() as session: print(session.run(tensor)) print(session.run(tensor[0])) print(session.run(tensor[1]))
```

Вывод:

```
[   1.45    -1. 0.2 102.1 ]

1.45

-1.0
```

Мы можем создать 2-й тензор или матрицу аналогичным образом:

Тема 18.6. Тензорные операции
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

В приведенном выше примере мы вводим несколько операций TensorFlow для векторов и матриц. Операции выполняют определенные вычисления на тензорах. Функции TensorFlow приведены в таблице ниже.

Оператор TensorFlow	Описание
tf.add	x + y
tf.subtract	xy
tf.multiply	x * y

Оператор TensorFlow	Описание
tf.div	x / y
tf.mod	x% y
tf.abs	| X |
tf.negative	-X
tf.sign	sign (x)
tf.square	x * x
tf.round	round(x)
tf.sqrt	SQRT (x)
tf.pow	x ^ y
tf.exp	e ^ x
tf.log	log (x)
tf.maximum	max (x, y)
tf.minimum	min (x, y)
tf.cos	cos (x)
tf.sin	sin (x)

Операции TensorFlow, перечисленные в таблице выше, работают с тензорными объектами и выполняются поэлементно. Поэтому, если вы хотите вычислить косинус для вектора x, операция TensorFlow будет выполнять вычисления для каждого элемента в переданном тензоре:

```
tensor_1d = np.array([0, 0, 0])

tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64) with tf.Session() as session:
print session.run(tf.cos(tensor)) Вывод:
[ 1. 1. 1.]
```

Тема 18.7. Матричные операции
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Матричные операции очень важны для моделей машинного обучения, таких как линейная регрессия, поскольку они часто используются в них. TensorFlow поддерживает все наиболее распространенные операции с матрицами, такие как умножение, инверсия, вычисление определителя, решение линейных уравнений и многое другое. Давайте напишем некоторый код, который будет выполнять базовые матричные операции, такие как умножение, транспонирование, вычислении определителя, умножения и другие.
Ниже приведены основные примеры вызова этих операций.

```
import tensorflow as tf import numpy as np

def convert(v, t=tf.float32):
return tf.convert_to_tensor(v, dtype=t)
m1 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m2 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m3 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m4 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m5 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m_tranpose = tf.transpose(m1)

m_mul = tf.matmul(m1, m2)
m_det = tf.matrix_determinant(m3) m_inv = tf.matrix_inverse(m4)
m_solve = tf.matrix_solve(m5, [[1], [1], [1], [1]])


with tf.Session() as session: print(session.run(m_tranpose)) print(session.run(m_mul)) print(session.run(m_inv)) print(session.run(m_det)) print(session.run(m_solve))
```

TensorFlow поддерживает различные виды редукции. Редукция - это операция, которая удаляет один или несколько измерений из тензора, выполняя определенные операции по этим измерениям. Мы представим несколько из них в приведенном ниже примере.

```
import tensorflow as tf import numpy as np
def convert(v, t=tf.float32):

return tf.convert_to_tensor(v, dtype=t) x = convert(
np.array( [ (1, 2, 3), (4, 5, 6), (7, 8, 9) ]), tf.int32)

bool_tensor = convert([(True, False, True), (False, False, True), (True, False, False)], tf.bool)
red_sum_0 = tf.reduce_sum(x)

red_sum = tf.reduce_sum(x, axis=1) red_prod_0 = tf.reduce_prod(x) red_prod = tf.reduce_prod(x, axis=1) red_min_0 = tf.reduce_min(x) red_min = tf.reduce_min(x, axis=1) red_max_0 = tf.reduce_max(x) red_max = tf.reduce_max(x, axis=1) red_mean_0 = tf.reduce_mean(x)
red_mean = tf.reduce_mean(x, axis=1) red_bool_all_0 = tf.reduce_all(bool_tensor) red_bool_all = tf.reduce_all(bool_tensor, axis=1) red_bool_any_0 = tf.reduce_any(bool_tensor) red_bool_any = tf.reduce_any(bool_tensor, axis=1) with tf.Session() as session:
print("Reduce sum without passed axis parameter: ", session.run(red_sum_0)) print("Reduce sum with passed axis=1: ", session.run(red_sum))
print("Reduce product without passed axis parameter: ", session.run(red_prod_0)) print("Reduce product with passed axis=1: ", session.run(red_prod)) print("Reduce min without passed axis parameter: ", session.run(red_min_0)) print("Reduce min with passed axis=1: ", session.run(red_min))


print("Reduce max without passed axis parameter: ", session.run(red_max_0)) print("Reduce max with passed axis=1: ", session.run(red_max))
print("Reduce mean without passed axis parameter: ", session.run(red_mean_0)) print("Reduce mean with passed axis=1: ", session.run(red_mean))
print("Reduce bool all without passed axis parameter: ", session.run(red_bool_all_0))
print("Reduce bool all with passed axis=1: ", session.run(red_bool_all))
```

Тема 18.8. Пример нейронной сети в TensorFlow
(Заочная: Лекционные занятия - 0,03ч.; Практические занятия - 0,09ч.; Самостоятельная работа - 0,9ч.; Очная: Лекционные занятия - 0,3ч.; Практические занятия - 0,3ч.; Самостоятельная работа - 0,6ч.)

Рассмотрим пример применения TensorFlow для создания простой трехслойной нейронной сети. В этом примере мы будем использовать набор данных MNIST (и связанный с ним загрузчик), который предоставляет пакет TensorFlow. Этот набор данных MNIST представляет собой набор изображений в оттенках серого размером 28 × 28 пикселей, которые представляют собой рукописные цифры. Он имеет 55 000 учебных рядов, 10 000 строк тестирования и 5000 строк проверки.

Мы можем загрузить данные, запустив:

```
import tensorflow as tf import numpy as np
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

One_hot=True - аргумент указывает, что вместо меток, связанных с каждым проецирование изображения, сама цифра, то есть «4», то есть вектор с
«один горячий» узел и все остальные узлы равно нулю, то есть [0, 0, 0, 0 , 1, 0, 0, 0, 0, 0]. Это позволяет нам легко подавать его в выходной слой нашей нейронной сети.
Затем мы можем настроить переменные-плейсхолдеры для данных обучения (и некоторые параметры обучения):

```
# Python optimisation variables

learning_rate = 0.5

epochs = 10

batch_size = 100

# declare the training data placeholders # input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])

# now declare the output data placeholder - 10 digits y = tf.placeholder(tf.float32, [None, 10])
```

Обратите внимание, что входной уровень X представляет содержит 784 узла, соответствующих 28 x 28 (= 784) пикселам, а выходной уровень Y- 10 узлов, соответствующих 10 возможных разрядов. Опять же, размер X равен (? X 784), где ? обозначает еще не заданное количество выборок, которые нужно ввести - это функция переменной-плейсхолдера.
Теперь нам нужно настроить переменные веса и смещения для трехслойной нейронной сети. Всегда должно быть L-1 количество тензоров веса / смещения, где L - количество слоев. Поэтому в этом случае нам нужно настроить два тензора для каждого:

```
# now declare the weights connecting the input to the hidden layer

W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1') b1 = tf.Variable(tf.random_normal([300]), name='b1')
# and the weights connecting the hidden layer to the output layer

W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')

b2 = tf.Variable(tf.random_normal([10]), name='b2')
```

В этом коде мы объявляем некоторые переменные для W1 и b1, веса и смещения для связей между входным и скрытым слоями. Эта нейронная сеть будет иметь 300 узлов в скрытом слое, поэтому размер весового тензора W1 равен [784, 300]. Мы инициализируем значения весов, используя случайное нормальное распределение со средним значением равным нулю и стандартным отклонением 0,03. TensorFlow имеет реплицированную версию случайной

6. Оценочные материалы текущего контроля

Раздел 1. НЕЙРОНЫ И ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. Что представляет из себя нейрон?

Нейрон представляет из себя элемент, который вычисляет выходной сигнал (по определенному правилу) из совокупности входных сигналов. То есть основная последовательность действий одного нейрона такая:

- Прием сигналов от предыдущих элементов сети
- Комбинирование входных сигналов
- Вычисление выходного сигнала
- Передача выходного сигнала следующим элементам нейронной сети

Раздел 2. ИСТОРИЯ НЕЙРОННЫХ СЕТЕЙ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. Какие основные этапы в истории исследования и применения искусственных нейронных сетей?

- 1943 — У. Маккалок и У. Питтс формализуют понятие нейронной сети в фундаментальной статье о логическом исчислении идей и нервной активности.
- 1948 — Н. Винер вместе с соратниками публикует работу о кибернетике. Основной идеей является представление сложных биологических процессов математическими моделями.
- 1949 — Д. Хебб предлагает первый алгоритм обучения.
- В 1958 Ф. Розенблатт изобретает однослойный перцептрон и демонстрирует его способность решать задачи классификации. Перцептрон обрёл популярность — его используют для распознавания образов, прогнозирования погоды и т. д.
- В 1960 году Уидроу совместно со своим студентом Хоффом на основе дельта-правила (формулы Уидроу) разработали Адалин, который сразу начал использоваться для задач предсказания и адаптивного управления.

Сейчас Адалин (адаптивный сумматор) является стандартным элементом многих систем обработки сигналов.

- В 1963 году в Институте проблем передачи информации АН СССР. А. П. Петровым проводится подробное исследование задач «трудных» для перцептрона.
- В 1969 году М. Минский публикует формальное доказательство ограниченности перцептрона и показывает, что он неспособен решать некоторые задачи (проблема «чётности» и «один в блоке»), связанные с инвариантностью представлений. Интерес к нейронным сетям резко спадает.
- В 1972 году Т. Кохонен и Дж. Андерсон независимо предлагают новый тип нейронных сетей, способных функционировать в качестве памяти.
- В 1973 году Б. В. Хакимов предлагает нелинейную модель с синапсами на основе сплайнов и внедряет её для решения задач в медицине, геологии, экологии.
- 1974 — Пол Дж. Вербос и А. И. Галушкин одновременно изобретают алгоритм обратного распространения ошибки для обучения многослойных перцептронов
- 1975 — Фукусима представляет когнитрон — самоорганизующуюся сеть, предназначенную для инвариантного распознавания образов, но это достигается только при помощи запоминания практически всех состояний образа.
- 1982 — после периода забвения, интерес к нейросетям вновь возрастает. Дж. Хопфилд показал, что нейронная сеть с обратными связями может представлять собой систему, минимизирующую энергию (так называемая сеть Хопфилда). Кохоненом представлены модели сети, обучающейся без учителя (нейронная сеть Кохонена), решающей задачи кластеризации, визуализации данных (самоорганизующаяся карта Кохонена) и другие задачи

предварительного анализа данных.

• 1986 — Дэвидом И. Румельхартом, Дж. Е. Хинтоном и Рональдом Дж. Вильямсом и одновременно с С. И. Барцевым и В. А. Охониным (Красноярская группа) переоткрыт и существенно развит метод обратного распространения ошибки. Начался взрыв интереса к обучаемым нейронным сетям.
• 2007 Джеффри Хинтоном в университете Торонто созданы алгоритмы глубокого обучения многослойных нейронных сетей. Успех обусловлен тем, что Хинтон при обучении нижних слоев сети использовал ограниченную машину Больцмана (RBM — Restricted Boltzmann Machine).

Раздел 3. КЛАССИФИКАЦИЯ НЕЙРОННЫХ СЕТЕЙ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. Классификация нейронных сетей

Классификация нейронных сетей по характеру обучения делит их на:
• нейронные сети, использующие обучение с учителем;
• нейронные сети, использующие обучение без учителя.

Нейронные сети, использующие обучение с учителем. Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар. Предъявляется выходной вектор, вычисляется выход сети и сравнивается с соответствующим целевым вектором. Далее веса изменяются в соответствии с алгоритмом, стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемого уровня.
Нейронные сети, использующие обучение без учителя. Обучение без учителя является намного более правдоподобной моделью обучения с точки зрения биологических корней искусственных нейронных сетей. Развитая Кохоненом и многими другими, она не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т. е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения, следовательно, выделяет статистические свойства обучающего множества и группирует сходные векторы в классы.

Классификация нейронных сетей по типу настройки весов делит их на:
• сети с фиксированными связями – весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи;
• сети с динамическими связями – для них в процессе обучения происходит настройка синаптических весов.

Классификация нейронных сетей по типу входной информации делит их на:
• аналоговые – входная информация представлена в форме действительных чисел;
• двоичные – вся входная информация в таких сетях представляется в виде нулей и единиц.

Раздел 4. АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ

В полносвязных нейронных сетях каждый нейрон передает свой выходной сигнал остальным нейронам, в том числе и самому себе. Все входные сигналы подаются всем нейронам. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких тактов функционирования сети.

В многослойных (слоистых) нейронных сетях нейроны объединяются в слои. Слой содержит совокупность нейронов с едиными входными сигналами. Число нейронов в слое может быть любым и не зависит от количества нейронов в других слоях. В общем случае сеть состоит из слоев, пронумерованных слева направо. Внешние входные сигналы подаются на входы нейронов входного
слоя (его часто нумеруют как нулевой), а выходами сети являются выходные

сигналы последнего слоя. Кроме входного и выходного слоев в многослойной нейронной сети есть один или несколько скрытых слоев. Связи от выходов нейронов некоторого слоя q к входам нейронов следующего слоя (q+1) называются последовательными.

Раздел 5. ФОРМАЛЬНЫЙ НЕЙРОН

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. Какие основные параметры нейрона?

У нейрона есть несколько входных каналов и только один выходной канал. По входным каналам на нейрон поступают данные задачи, а на выходе формируется результат работы. Нейрон вычисляет взвешенную сумму входных сигналов, а затем преобразует полученную сумму с помощью заданной нелинейной функции. Множество, состоящее из порогового уровня и всех весов, называют параметрами нейрона.

Здесь введены следующие обозначения: X1,X2, …, Xn - входной сигнал (паттерн), w1, w2,…,wn – весовые коэффициенты, b- порог нейрона

Сначала нейрон вычисляет взвешенную сумму

S □ □wi Xi i

□ b , далее

применяя функцию активации F(S) вычисляет выходной сигнал Y.

Функция активации нейрона - это функция, которая вычисляет выходной сигнал нейрона. На вход этой функции подается сумма всех произведений сигналов и весов этих сигналов.

Рассмотрим наиболее часто используемые функции активации.

а) Пороговая функция. Это простая кусочно-линейная функция. Если входное значение меньше порогового, то значение функции активации равно минимальному допустимому, иначе – максимально допустимому.

б) Линейный порог. Это несложная кусочно-линейная функция. Имеет два линейных участка, где функция активации тождественно равна минимально допустимому и максимально допустимому значению и есть участок, на котором функция строго монотонно возрастает.

в) Сигмоидальная функция или сигмоида (sigmoid). Это монотонно возрастающая дифференцируемая S-образная нелинейная функция. Сигмоида позволяет усиливать слабые сигналы и не насыщаться от сильных сигналов.

г) Гиперболический тангенс (hyperbolic tangent, tanh). Эта функция принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от –1 до 1. Подобно сигмоиде,

гиперболический тангенс может насыщаться. Однако, в отличие от сигмоиды, выход данной функции центрирован относительно нуля.

Недостатки формального нейрона:

- Предполагается, что нейрон мгновенно вычисляет свой выход, поэтому с помощью таких

нейронов нельзя моделировать непосредственно системы с внутренним состоянием.
• Формальные нейроны, в отличие от биологических, не могут обрабатывать информацию синхронно.
• Нет четких алгоритмов выбора функции активации.
• Невозможно регулировать работу всей сети.
• Излишняя формализация понятий «порог» и «весовые коэффициенты». У реальных нейронов порог меняется динамически, в зависимости от активности нейрона и общего состояния сети, а весовые коэффициенты изменяются в зависимости от проходящих сигналов.

Раздел 6. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. ОДНОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

Один нейрон может выполнять простейшие вычисления, но основные функции нейросети обеспечиваются не отдельными нейронами, а

соединениями между ними. Однослойный перцептрон представляет собой простейшую сеть, которая состоит из группы нейронов, образующих слой. Входные данные кодируются вектором значений, каждый элемент подается на соответствующий вход каждого нейрона в слое. В свою очередь, нейроны вычисляют выход независимо друг от друга. Размерность выхода (то есть количество элементов) равна количеству нейронов, а количество синапсов у всех нейронов должно быть одинаково и совпадать с размерностью входного сигнала.

Здесь X1,X2,X3- называется входной паттерн, Y1,Y2,Y3- выходной паттерн, а wi,j- это j-ый весовой коэффициент i-го нейрона

Раздел 7. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

Обучение нейронной сети - это процесс, в котором параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки параметров. Различают алгоритмы обучения с учителем и без учителя. Процесс обучения с учителем представляет собой предъявление сети выборки обучающих примеров. Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети.

Для того, чтобы нейронная сети была способна выполнить поставленную задачу, ее необходимо обучить. Различают алгоритмы обучения с учителем и без учителя. Процесс обучения с учителем представляет собой предъявление сети выборки обучающих примеров. Каждый образец подается на входы сети, затем проходит обработку внутри структуры НС, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети. Затем по определенному правилу вычисляется ошибка, и происходит изменение весовых коэффициентов связей внутри сети в зависимости от выбранного алгоритма. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки и веса подстраиваются для каждого вектора до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня.

При обучении без учителя обучающее множество состоит лишь из входных векторов. Обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы, т.е. чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения, следовательно, выделяет статистические свойства обучающего

множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения

невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения. Это не является серьезной проблемой. Обычно не сложно идентифицировать связь между входом и выходом, установленную сетью. Для обучения нейронных сетей без учителя применяются сигнальные метод обучения Хебба и Ойа.

Математически процесс обучения можно описать следующим образом. В процессе функционирования нейронная сеть формирует выходной сигнал Y, реализуя некоторую функцию Y = G(X). Если архитектура сети задана, то вид функции G определяется значениями синаптических весов и смещенной сети.

Пусть решением некоторой задачи является функция Y = F(X), заданная параметрами входных-выходных данных (X1, Y1), (X2, Y2), …, (XN, YN), для которых Yk = F(Xk) (k = 1, 2, …, N).

Обучение состоит в поиске (синтезе) функции G, близкой к F в смысле некоторой функции ошибки E.

Если выбрано множество обучающих примеров – пар (XN, YN) (где k = 1, 2, …, N) и способ вычисления функции ошибки E, то обучение нейронной сети превращается в задачу многомерной оптимизации, имеющую очень большую размерность, при этом, поскольку функция E может иметь произвольный вид обучение в общем случае – многоэкстремальная невыпуклая задача оптимизации.

Для решения этой задачи могут использоваться следующие (итерационные) алгоритмы:

1. алгоритмы локальной оптимизации с вычислением частных производных первого порядка:
- градиентный алгоритм (метод наискорейшего спуска),
- методы с одномерной и двумерной оптимизацией целевой функции в направлении антиградиента,
- метод сопряженных градиентов,

- методы, учитывающие направление антиградиента на нескольких шагах алгоритма;

2. алгоритмы локальной оптимизации с вычислением частных производных первого и второго порядка:
- метод Ньютона,
- методы оптимизации с разреженными матрицами Гессе,
- квазиньютоновские методы,
- метод Гаусса-Ньютона,
- метод Левенберга-Марквардта и др.;

3. стохастические алгоритмы оптимизации:
- поиск в случайном направлении,
- имитация отжига,
- метод Монте-Карло (численный метод статистических испытаний);

4. алгоритмы глобальной оптимизации (задачи глобальной оптимизации решаются с помощью перебора значений переменных, от которых зависит целевая функция).

Раздел 8. МНОГОСЛОЙНАЯ НЕЙРОННАЯ СЕТЬ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. МНОГОСЛОЙНАЯ НЕЙРОННОЙ СЕТЬ

Многослойная нейронная сеть (персептрон) — это нейронная сеть, состоящая из входного, выходного и расположенных между ними одного (или нескольких) скрытых слоев нейронов.

Чтобы построить многослойный персептрон, необходимо выбрать его параметры по следующему алгоритму:
- Определить, какой смысл вкладывается в компоненты входного вектора

X. Входной вектор должен содержать формализованное условие задачи, то есть всю информацию, необходимую для того, чтобы получить ответ.

• Выбрать выходной вектор Y таким образом, чтобы его компоненты содержали полный ответ для поставленной задачи.
• Выбрать вид функции активации нейронов. При этом желательно учесть специфику задачи, так как удачный выбор увеличит скорость обучения.
• Выбрать количество слоев и нейронов в слое.
• Задать диапазон изменения входов, выходов, весов и пороговых уровней на основе выбранной функции активации.
• Присвоить начальные значения весам и пороговым уровням. Начальные значения не должны быть большими, чтобы нейроны не оказались в насыщении (на горизонтальном участке функции активации), иначе обучение будет очень медленным. Начальные значения не должны быть и слишком малыми, чтобы выходы большей части нейронов не были равны нулю, иначе обучение тоже замедлится.
• Провести обучение, то есть подобрать параметры сети так, чтобы задача решалась наилучшим образом. По окончании обучения сеть сможет решать задачи того типа, которым она обучена.
• Подать на вход сети условия задачи в виде вектора X. Рассчитать выходной вектор Y, который и даст формализованное решение задачи.

Раздел 9. ВВЕДЕНИЕ В KERAS И ЕГО ОСНОВНЫЕ ПРИНЦИПЫ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. ВВЕДЕНИЕ В KERAS И ЕГО ОСНОВНЫЕ ПРИНЦИПЫ

В современном мире, начиная со здравоохранения и заканчивая мануфактурным производством, повсеместно используется глубинное обучение. Компании обращаются к этой технологии для решения сложных проблем, таких как распознавание речи и объектов, машинный перевод и так далее.

Одним из самых впечатляющих достижений этого года был AlphaGo, обыгравший лучшего в мире игрока в го. Кроме как в го, машины обошли людей и в других играх: шашки, шахматы, реверси, и джеопарди.
Возможно, победа в настольной игре кажется неприменимой в решении реальных проблем, однако это совсем не так. Го был создан так, чтобы в нем не мог победить искусственный интеллект. Для этого ему необходимо было бы научиться одной важной для этой игры вещи – человеческой интуиции. Теперь с помощью данной разработки возможно решить множество проблем, недоступных компьютеру раньше.
Очевидно, глубинное обучение еще далеко от совершенства, но оно уже близко к тому, чтобы приносить коммерческую пользу. Например, эти самоуправляемые машины. Известные компании вроде Google, Tesla и Uber уже пробуют внедрить автономные автомобили на улицы города. Ford предсказывает значительное увеличение доли беспилотных транспортных средств уже к 2021 году. Правительство США также успело разработать для них свод правил безопасности.
Keras является высокоуровневыми нейронными сетями API, написанный на Python и могут работать поверх TensorFlow, CNTK или Теано. Он был разработан с упором на возможность быстрого экспериментирования. Способность идти от идеи к результату с наименьшей возможной задержкой является ключом к проведению хороших исследований.

Раздел 10. МОДЕЛИ KERAS

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. МОДЕЛИ KERAS

Основная структура данных Keras - это модель, способ организации слоев. В Keras доступны два основных типа моделей: последовательная модель Sequential и класс Model, используемый с функциональным API. Простейшим типом модели является Sequential

модель, которая представляет собой линейную совокупность слоев. Для более сложных архитектур необходимо использовать функциональный API Keras, который позволяет создавать произвольные графики слоев.

Раздел 11. СЛОИ В KERAS

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. СЛОИ В KERAS

Все слои Keras имеют ряд общих методов:

- layer.get_weights()- возвращает веса слоя в виде списка массивов Numpy.
- layer.set_weights(weights)- устанавливает веса слоя из списка массивов (с теми же формами, что и выход get_weights).
- layer.get_config() - возвращает словарь, содержащий конфигурацию слоя.

Слой может быть восстановлен из его конфигурации используя следующий:

```
layer = Dense(32)

config = layer.get_config()
reconstructed_layer = Dense.from_config(config)
```

Если слой имеет один узел (т. е. если он не является общим слоем), то можно получить его входной тензор, выходной тензор, размерность входного массива и размерность выходного массива через свойства:

- layer.input
- layer.output
- layer.input_shape
- layer.output_shape

Раздел 12. ОСНОВЫ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОЙ МОДЕЛЬЮ KERAS

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. ОСНОВЫ РАБОТЫ С ПОСЛЕДОВАТЕЛЬНОЙ МОДЕЛЬЮ KERAS

Создать последовательную модель (Sequential модель) можно передав список экземпляров слоя в конструктор класса Sequential:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential([Dense(32, input_shape=(784,)), Activation('relu'), Dense(10), Activation('softmax'),])
```

Также можно просто добавить слои с помощью метода add(). model = Sequential()
model.add(Dense(32, input_dim=784)) model.add(Activation('relu'))

Раздел 13. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В KERAS

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ В KERAS

Сверточные нейронные сети являются одной из форм многослойных нейронных сетей. Здесь приведена схема типичного CNN. Первая часть состоит

из слоев свертки и максимального пула, которые выступают в качестве экстрактора признаков. Вторая часть состоит из полносвязного слоя, который выполняет нелинейные преобразования извлеченных признаков и действует как классификатор.

На приведенной выше диаграмме вход подается в сеть последовательных слоев Conv, Pool и Dense. Выходной сигнал может быть слоем softmax, указывающим, есть ли кошка или что-то

еще. Также, а качестве выходного может быть использован сигмоидный слой, на выходе которого будет вероятность того, что изображение будет кошкой. Рассмотрим слои более подробно.
Сверточный слой можно рассматривать как глаза сверточной нейронной сети. Нейроны в этом слое ищут определенные особенности. Свертку можно рассматривать как взвешенную сумму между двумя сигналами или функциями. Пример операции свертки на матрице размером 5 × 5 с ядром размером 3 × 3 показан ниже. Ядро свертки скользит по всей матрице для

получения карты активации.

Предположим, что входное изображение имеет размер 32x32x3, т.е. это трехмерный массив глубины 3. Любой фильтр свертки, который мы определяем на этом слое, должен иметь глубину, равную глубине ввода. Поэтому мы можем выбрать фильтры свертки глубины 3 (например, 3x3x3 или 5x5x3 или 7x7x3 и т. Д.). Выберем фильтр свертки размера 3x3x3, т.е сверточное ядро будет кубом вместо квадрата.
Если мы сможем выполнить операцию свертки, сдвинув фильтр 3x3x3 на все изображение размером 32x32x3, то мы получим изображение с разрешением 30x30x1. Это связано с тем, что операция свертки невозможна для полосы шириной 2 пикселя вокруг изображения. Фильтр всегда находится внутри изображения и поэтому 1 пиксель удаляется от левой, правой, верхней и нижней части изображения.
Для входного изображения 32x32x3 и размера фильтра 3x3x3 у нас есть 30x30x1 местоположения, и для каждого местоположения существует нейрон. Тогда выходы 30x30x1 или активации всех нейронов называются

картами активации. Карта активации одного уровня служит входом для следующего слоя.
В нашем примере есть 30 × 30 = 900 нейронов, потому что есть много мест, где может применяться фильтр 3x3x3. В отличие от традиционных нейронных сетей, где веса и смещения нейронов независимы друг от друга, в случае сверточных нейронных сетей, нейроны, соответствующие одному фильтру в слое, имеют одинаковые веса и смещения. В приведенном выше случае мы сдвигаем окно на 1 пиксель за раз. Мы также можем сдвинуть окно более чем на 1 пиксель. Это число называется шагом.
Как правило, используют более одного фильтра в одном слое свертки. Если мы используем 32 фильтра, у нас будет карта активации размером 30x30x32.
Обратите внимание, что все нейроны, связанные с одним и тем же фильтром, имеют одинаковые веса и смещения. Таким образом, количество весов при использовании 32 фильтров - это 3x3x3x32 = 288, а число смещений - 32.
На картинке показаны 32 карты активации, полученные от применения

сверточных ядер.
Как вы можете видеть, после каждой свертки результат уменьшается по размеру (так как в этом случае мы переходим от 32 × 32 до 30 × 30). Для удобства стандартная практика заключается в том, чтобы накладывать нули на границу входного слоя таким образом, чтобы выход был такого же размера, как и входной. Итак, в этом примере, если мы добавим дополнение размером 1 по

обе стороны от входного слоя, размер выходного уровня будет 32x32x32, что упростит реализацию.
Рассмотрим, как сверточные нейронные сети анализируют изображения.

На приведенном выше рисунке большие квадраты указывают область, в которой выполняется операция свертки, а малые квадраты указывают выход операции, которая является просто числом. Следует отметить следующие замечания:

- В первом слое квадрат, обозначенный 1, получается из области изображения, на которой

окрашены листья.

• Во втором слое квадрат с меткой 2 получается из большего квадрата в первом слое. Числа в этом квадрате получены из нескольких областей из входного изображения. В частности, вся площадь вокруг левого уха кошки отвечает за значение на квадрате, отмеченном 2.

• Аналогично, в третьем слое этот каскадный эффект приводит к тому, что квадрат, обозначенный 3, получается из большой области вокруг области ноги.

Из сказанного выше можно сказать, что начальные слои анализируют более мелкие области изображения и, следовательно, могут обнаруживать только простые признаки, такие как края / углы и т. д. По мере того как мы идем глубже в сеть, нейроны получают информацию из более крупных частей изображения и от различных других нейронов. Таким образом, нейроны на более поздних слоях могут изучить более сложные функции, такие как глаза, ноги, и т.д

Слой пулинга в основном используется сразу после сверточного слоя для уменьшения пространственного размера (только по ширине и высоте, а не по глубине). Это уменьшает количество параметров, поэтому вычисление уменьшается. Использование меньшего количества параметров позволяет избежать переобучения. Переобучение - это условие, когда обученная модель отлично работает с данными обучения, но не очень хорошо работает в тестовых данных.

Наиболее распространенной формой пулинга является максимальный пулинг, в котором мы берем фильтр размера и применяем максимальную операцию max с определенной частью изображения.

На рисунке показан максимальный пул с размером фильтра 2 × 2 и шагом

2. Выход представляет собой максимальное значение в области 2 × 2, показанной с использованием окруженных цифр. Наиболее распространенная операция пулинга выполняется с фильтром размером 2 × 2 с шагом 2. Это существенно уменьшает размер ввода на половину.

Раздел 14. РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. РАСПОЗНАВАНИЕ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

MNIST - это набор данных, разработанный Янном ЛеКуном, Коринной Кортес и Кристофер Бургес для оценки моделей машинного обучения по проблеме классификации рукописных цифр. Набор данных был построен из ряда отсканированных наборов документов, доступных в Национальном институте стандартов и технологий (NIST). Изображения цифр были взяты из множества отсканированных документов, нормированных по размеру и по центру. Это делает его отличным набором данных для оценки моделей, позволяя разработчику сосредоточиться на механизме обучения с очень небольшой очисткой данных или необходимой подготовкой.

Каждое изображение представляет собой квадрат размером 28 на 28 пикселей (всего 784 пикселя). В этом наборе 60 000 изображений используются для обучения модели, и для ее тестирования используется отдельный набор из

10 000 изображений. Это задача распознавания 10 цифр (от 0 до 9) или классификация на 10 классов.

Библиотека глубокого обучения Keras предоставляет удобный метод mnist.load_data() для загрузки набора данных MNIST. Набор данных загружается автоматически при первом вызове этой функции и сохраняется в вашем домашнем каталоге в ~ / .keras /datasets /mnist.pkl.gz в виде файла 15 МБ. Это очень удобно для разработки и тестирования моделей глубокого обучения.

Чтобы продемонстрировать, насколько легко загружать набор данных MNIST, мы сначала напишем небольшой скрипт для загрузки и визуализации первых четырех изображений в

наборе учебных материалов.
from keras.datasets import mnist import matplotlib.pyplot as plt
# load (downloaded if needed) the MNIST dataset (X_train, y_train), (X_test, y_test) = mnist.load_data() # plot 4 images as gray scale
plt.subplot(221)

plt.imshow(X_train[0], cmap=plt.get_cmap('gray')) plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray')) plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray')) plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray')) # show the plot
plt.show()
Запустив приведенный выше пример, вы должны увидеть изображение ниже.

Раздел 15. ПРЕДСТАВЛЕНИЕ СЛОВ В ВЕКТОРНОМ ПРОСТРАНСТВЕ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. ПРЕДСТАВЛЕНИЕ СЛОВ В ВЕКТОРНОМ ПРОСТРАНСТВЕ

Векторизация слов (world embeding) - это класс подходов для представления слов и документов с использованием векторного представления. Это улучшение по сравнению с традиционными схемами кодирования, где для представления каждого слова использовались большие разреженные векторы или оценка каждого слова в векторе для представления целого словарного запаса. Эти представления были скудными, потому что словари были обширными, и данное слово или документ представлялось бы большим вектором, состоящим в основном из нулевых значений.
Вместо этого в world embeding слова представлены плотными векторами, где вектор представляет проекцию слова в непрерывное векторное пространство.

Представление слова в векторном пространстве получается из текста и основывается на словах, которые окружают слово, когда оно используется.
Два популярных примера методов вложения слов в текст включают:

- Word2Vec.
- GloVe.

В дополнение к этим ранее разработанным методам, векторизацию слов можно изучить как часть модели глубокого обучения.

Раздел 16. LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ

LSTM НЕЙРОННЫЕ СЕТИ И ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ

Раздел 17. КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ

КЛАССИФИКАЦИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ С ИСПОЛЬЗОВАНИЕМ LSTM НЕЙРОННЫХ СЕТЕЙ

Раздел 18. НЕЙРОННЫЕ СЕТИ НА ОСНОВЕ БИБЛИОТЕКИ TENSORFLOW

Форма контроля/оценочное средство: Задача

Вопросы/Задания:

1. НЕЙРОННЫЕ СЕТИ НА ОСНОВЕ БИБЛИОТЕКИ TENSORFLOW

18.1. Начало работы с TensorFlow

TensorFlow - это библиотека программного обеспечения с открытым исходным кодом, созданная Google, которая используется для внедрения систем машинного обучения и глубокого обучения. Эти два имени содержат ряд мощных алгоритмов, которые разделяют общую задачу - позволить компьютеру узнать, как автоматически определять сложные шаблоны и / или

принимать наилучшие возможные решения. TensorFlow, в основе своей, является библиотекой для программирования потока данных. Он использует различные методы оптимизации, чтобы сделать вычисления математических выражений проще и эффективнее.
Некоторые из ключевых особенностей TensorFlow:
• Эффективно работает с математическими выражениями, включающими многомерные массивы
• Хорошая поддержка глубоких нейронных сетей и концепций машинного обучения
• Использование GPU / CPU, где один и тот же код может быть выполнен на обеих архитектурах
• Высокая масштабируемость вычислений на машинах и огромные массивы данных
Установить TensorFlow можно выполнив команду:

Загрузка и сборка TensorFlow может занять несколько минут.

18.2. Основы работы в TensorFlow

В TensorFlow вычисление описывается с использованием графов потоков данных. Каждый узел графа представляет собой экземпляр математической операции (например, сложение, деление или умножение), и каждое ребро представляет собой многомерный набор данных (тензор), на котором выполняются операции.

Прежде чем перейти к обсуждению элементов TensorFlow, мы сначала создадим сеанс работы с TensorFlow, чтобы понять, как выглядит программа TensorFlow.
В TensorFlow константы создаются с использованием функции: constant(value, dtype=None, shape=None, name='Const', verify_shape=False), где value постоянное значение, которое будет использоваться при дальнейших вычислениях, dtype является параметром, указывающим тип данных (например, float32/64, int8/16), shape является необязательным параметром, указывающим размер массива данных, name является необязательным задающим имя для тензора. Если вам нужны константы с определенными значениями внутри вашей обучающей модели, тогда объект типа constant может использоваться как в следующем примере:
z = tf.constant(5.2, name="x", dtype=tf.float32)
Переменные в TensorFlow являются буферами в памяти, содержащими тензоры, которые должны быть явно инициализированы. Просто вызывая конструктор, мы добавляем переменную в вычислительный граф. Переменные

особенно полезны, как только вы начинаете с моделей обучения, и они используются для хранения и обновления параметров. Начальное значение, переданное в качестве аргумента конструктора, представляет собой тензор или объект, который может быть преобразован или возвращен как тензор. Это означает, что если мы хотим заполнить переменную некоторыми предопределенными или случайными значениями, которые будут использоваться впоследствии в процессе обучения и обновлены при итерациях, мы можем определить ее следующим образом:
k = tf.Variable(tf.zeros([1]), name="k")
Другой способ использования переменных в TensorFlow - это вычисления, когда эта

переменная не является обучаемой и может быть определена следующим образом:
k = tf.Variable(tf.add(a, b), trainable= False)

Сеанс TensorFlow инкапсулирует управление и состояние среды выполнения TensorFlow. Сеанс без параметров будет использовать граф по умолчанию, созданный в текущем сеансе, иначе класс сеанса принимает параметр графа, который используется в этом сеансе для выполнения. Ниже приведен краткий фрагмент кода, в котором показано, как термины, определенные выше, могут использоваться в TensorFlow для вычисления простой линейной функции y=a*x+b:
import tensorflow as tf
x = tf.constant(-2.0, name="x", dtype=tf.float32) a = tf.constant(5.0, name="a", dtype=tf.float32) b = tf.constant(13.0, name="b", dtype=tf.float32) y = tf.Variable(tf.add(tf.multiply(a, x), b))
init = tf.global_variables_initializer() with tf.Session() as session:
session.run(init) print(session.run(y))

18.3. Определение вычислительных графов в TensorFlow

Преимущество при работе с графами потоков данных заключается в том, что модель отделена от ее исполнения, т.е. неважно на каком вычислительном устройстве выполняется программный код, на процессоре, графическом процессоре или некоторой комбинации. Программа в среде TensorFlow может выполняться на процессоре или графическом процессоре, а вся сложность, связанная с кодом исполнения скрыта от пользователя. Граф вычисления - это встроенный процесс, который использует библиотеку, не требуя прямого вызова объекта графа. Граф вычислений может быть построен в процессе использования библиотеки TensorFlow без необходимости явно создавать объекты Graph.
Объект Graph в TensorFlow может быть создан в результате простой строки кода c = tf.add(a, b). Это код создаст операционный узел, который принимает два тензора a и b которые вычисляют их сумму c в качестве результата.
Плейсхолдер - это способ, позволяющий разработчикам вводить данные в график вычислений через заполнители, которые связаны внутри некоторых выражений. Сигнатура плейсхолдера:
placeholder(dtype, shape= None , name= None)
где dtype - тип элементов в тензорах.
Преимущество плейсхолдеров заключается в том, что они позволяют разработчикам создавать операции в вычислительном графе вообще, без необходимости заранее предоставлять данные, и данные могут быть добавлены во время выполнения из внешних источников.
Рассмотрим простую задачу умножения двух целых чисел x и y в TensorFlow и использованием плейсхолдера:
import tensorflow as tf
x = tf.placeholder(tf.float32, name="x") y = tf.placeholder(tf.float32, name="y")

z = tf.multiply(x, y, name="z") with tf.Session() as session:
print(session.run(z, feed_dict={x: 2.1, y: 3.0}))

18.4. Визуализация вычислительного графа с помощью TensorBoard

TensorBoard - это инструмент визуализации для анализа графиков потока данных. Он может быть полезно для лучшего понимания моделей машинного обучения. С TensorBoard вы можете получить представление о различных типах статистики о параметрах и подробностях о частях вычислительного графа. Глубокая нейронная сеть имеет большое количество узлов. TensorBoard позволяет разработчикам получить представление о каждом узле и о том, как вычисление выполняется во время выполнения TensorFlow.
Теперь давайте вернемся к нашему примеру где мы определили линейную функцию y = a*x + b.
Чтобы регистрировать события с сеанса, которые позже могут использоваться в TensorBoard, TensorFlow предоставляет класс FileWriter. Его можно использовать для создания

файла событий для хранения сводок и событий. Конструктор FileWriter принимает шесть параметров и выглядит так:

__init__(logdir, graph=None, max_queue=10, flush_secs=120, graph_def=None, filename_suffix=None) - где требуется указать обязательный параметр logdir, а другие - значения по умолчанию. Параметр графа будет передан из объекта сеанса, созданного в программе. Полный код примера выглядит так:

```
import tensorflow as tf
x = tf.constant(-2.0, name="x", dtype=tf.float32) a = tf.constant(5.0, name="a", dtype=tf.float32) b = tf.constant(13.0, name="b", dtype=tf.float32) y = tf.Variable(tf.add(tf.multiply(a, x), b))
init = tf.global_variables_initializer()

with tf.Session() as session:
merged = tf.summary.merge_all()
writer = tf.summary.FileWriter("logs", session.graph) session.run(init)
print(session.run(y))
```

Мы добавили две новые строки в которых создается и используется объект FileWriter для вывода событий в файл, как описано выше.

После запуска программы у нас появляется файл в журналах каталогов, и для того чтобы посмотреть содержимое этого файла необходимо запустить tensorboard:

```
tensorboard --logdir logs/
```

После открытия http://localhost:6006 и нажатия на пункт меню «Графики»

(расположенный в верхней части страницы) вы сможете увидеть график, как на картинке ниже:

TensorBoard маркирует константы и сводные узлы конкретными символами, которые показаны ниже.

99

18.5. Математика с TensorFlow

Тензоры - это основные структуры данных в TensorFlow, и они представляют собой соединительные грани в графе потока данных.

Тензор просто идентифицирует многомерный массив или список. Тензорную структуру можно идентифицировать тремя параметрами: рангом, размером и типом.

- Ранг: Определяет количество измерений тензора. Ранг известен как порядок или n-размерности тензора, где, например, тензор ранга 1 является тензором вектора или ранга 2, является матрицей.
- Размер: Размер тензора - это количество строк и столбцов.
- Тип: Тип данных элементов тензора.

Чтобы построить тензор в TensorFlow, мы можем построить n-мерный массив. Это можно сделать легко, используя библиотеку NumPy, или путем преобразования n-мерного массива Python в тензор TensorFlow.

Чтобы построить 1-мерный тензор, мы будем использовать массив NumPy:

```
import numpy as np
tensor_1d = np.array([1.45, -1, 0.2, 102.1])
```

Работа с подобным массивом аналогична работе со встроенным списком Python. Основное

отличие состоит в том, что массив NumPy также содержит некоторые дополнительные свойства, такие как размер, форма и тип.
>> print tensor_1d
[1.45 -1. 0.2 102.1]
>> print tensor_1d[0] 1.45
>> print tensor_1d[2] 0.2
>> print tensor_1d.ndim 1
>> print tensor_1d.shape

(4,)
>> print tensor_1d.dtype float64

Массив NumPy можно легко преобразовать в тензор TensorFlow используя вспомогательную функцию convert_to_tensor, что помогает разработчикам преобразовывать объекты Python в объекты тензора. Эта функция принимает тензорные объекты, массивы NumPy и списки Python.
tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)
Теперь, если мы привяжем наш тензор к сеансу TensorFlow, мы сможем увидеть результаты нашего преобразования.
import numpy as np import tensorflow as tf
tensor_1d = np.array([1.45, -1, 0.2, 102.1])

tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)

with tf.Session() as session: print(session.run(tensor)) print(session.run(tensor[0])) print(session.run(tensor[1]))
Вывод:

[1.45 -1. 0.2 102.1]

1.45

-1.0

Мы можем создать 2-й тензор или матрицу аналогичным образом:

tensor_2d = np.array(np.random.rand(4, 4), dtype='float32') tensor_2d_1 = np.array(np.random.rand(4, 4), dtype='float32') tensor_2d_2 = np.array(np.random.rand(4, 4), dtype='float32')

m1 = tf.convert_to_tensor(tensor_2d) m2 = tf.convert_to_tensor(tensor_2d_1) m3 = tf.convert_to_tensor(tensor_2d_2) mat_product = tf.matmul(m1, m2) mat_sum = tf.add(m2, m3) mat_det = tf.matrix_determinant(m3)

with tf.Session() as session:
print session.run(mat_product) print session.run(mat_sum) print session.run(mat_det)
18.6. Тензорные операции

В приведенном выше примере мы вводим несколько операций TensorFlow для векторов и матриц. Операции выполняют определенные вычисления на тензорах. Функции TensorFlow приведены в таблице ниже.
Оператор TensorFlow Описание

tf.add
x + y
tf.subtract
xy
tf.multiply
x * y

Оператор TensorFlow Описание
tf.div
x / y
tf.mod
x% y
tf.abs
| X |
tf.negative
-X
tf.sign
sign (x)
tf.square
x * x
tf.round
round(x)
tf.sqrt
SQRT (x)
tf.pow
x ^ y
tf.exp
e ^ x
tf.log
log (x)
tf.maximum
max (x, y)
tf.minimum
min (x, y)
tf.cos
cos (x)
tf.sin
sin (x)

Операции TensorFlow, перечисленные в таблице выше, работают с тензорными объектами и выполняются поэлементно. Поэтому, если вы хотите вычислить косинус для вектора x, операция TensorFlow будет выполнять вычисления для каждого элемента в переданном тензоре:
tensor_1d = np.array([0, 0, 0])

tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64) with tf.Session() as session:
print session.run(tf.cos(tensor)) Вывод:
[1. 1. 1.]

18.7. Матричные операции

Матричные операции очень важны для моделей машинного обучения, таких как линейная регрессия, поскольку они часто используются в них. TensorFlow поддерживает все наиболее распространенные операции с матрицами, такие как умножение, инверсия, вычисление определителя, решение линейных уравнений и многое другое. Давайте напишем некоторый код, который будет выполнять базовые матричные операции, такие как умножение,

транспонирование, вычислении определителя, умножения и другие.
Ниже приведены основные примеры вызова этих операций.
import tensorflow as tf import numpy as np

def convert(v, t=tf.float32):
return tf.convert_to_tensor(v, dtype=t)
m1 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m2 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m3 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m4 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m5 = convert(np.array(np.random.rand(4, 4), dtype='float32')) m_tranpose = tf.transpose(m1)

m_mul = tf.matmul(m1, m2)
m_det = tf.matrix_determinant(m3) m_inv = tf.matrix_inverse(m4)
m_solve = tf.matrix_solve(m5, [[1], [1], [1], [1]])

with tf.Session() as session: print(session.run(m_tranpose)) print(session.run(m_mul)) print(session.run(m_inv)) print(session.run(m_det)) print(session.run(m_solve))

TensorFlow поддерживает различные виды редукции. Редукция - это операция, которая удаляет один или несколько измерений из тензора, выполняя определенные операции по этим измерениям. Мы представим несколько из них в приведенном ниже примере.
import tensorflow as tf import numpy as np
def convert(v, t=tf.float32):

return tf.convert_to_tensor(v, dtype=t) x = convert(
np.array([(1, 2, 3), (4, 5, 6), (7, 8, 9)]), tf.int32)

bool_tensor = convert([(True, False, True), (False, False, True), (True, False, False)], tf.bool)
red_sum_0 = tf.reduce_sum(x)

red_sum = tf.reduce_sum(x, axis=1) red_prod_0 = tf.reduce_prod(x) red_prod = tf.reduce_prod(x, axis=1) red_min_0 = tf.reduce_min(x) red_min = tf.reduce_min(x, axis=1) red_max_0 = tf.reduce_max(x) red_max = tf.reduce_max(x, axis=1) red_mean_0 = tf.reduce_mean(x)
red_mean = tf.reduce_mean(x, axis=1) red_bool_all_0 = tf.reduce_all(bool_tensor) red_bool_all = tf.reduce_all(bool_tensor, axis=1) red_bool_any_0 = tf.reduce_any(bool_tensor) red_bool_any = tf.reduce_any(bool_tensor, axis=1) with tf.Session() as session:
print("Reduce sum without passed axis parameter: ", session.run(red_sum_0)) print("Reduce sum with passed axis=1: ", session.run(red_sum))
print("Reduce product without passed axis parameter: ", session.run(red_prod_0)) print("Reduce product with passed axis=1: ", session.run(red_prod)) print("Reduce min without passed axis parameter: ", session.run(red_min_0)) print("Reduce min with passed axis=1: ", session.run(red_min))

print("Reduce max without passed axis parameter: ", session.run(red_max_0)) print("Reduce max with passed axis=1: ", session.run(red_max))
print("Reduce mean without passed axis parameter: ", session.run(red_mean_0)) print("Reduce mean with passed axis=1: ", session.run(red_mean))
print("Reduce bool all without passed axis parameter: ", session.run(red_bool_all_0))
print("Reduce bool all with passed axis=1: ", session.run(red_bool_all))

print("Reduce bool any without passed axis parameter: ", session.run(red_bool_any_0))
print("Reduce bool any with passed axis=1: ", session.run(red_bool_any))

Вывод программы:

Reduce sum without passed axis parameter: 45 Reduce sum with passed axis=1: [6 15 24]
Reduce product without passed axis parameter: 362880 Reduce product with passed axis=1: [6 120 504] Reduce min without passed axis parameter: 1
Reduce min with passed axis=1: [1 4 7] Reduce max without passed axis parameter: 9 Reduce max with passed axis=1: [3 6 9]
Reduce mean without passed axis parameter: 5

Reduce mean with passed axis=1: [2 5 8]

Reduce bool all without passed axis parameter: False Reduce bool all with passed axis=1: [False False False] Reduce bool any without passed axis parameter: True Reduce bool any with passed axis=1: [True True True]
Первым параметром операторов редукции является тензор, который мы хотим уменьшить. Второй параметр - это индексы размерностей, по которым мы хотим выполнить редукцию. Этот параметр является необязательным, и, если его не передать, редукция будет выполняться по всем измерениям.
Сегментация - это процесс, в котором одним из измерений является процесс отображения размеров на предоставленные сегментные индексы, а результирующие элементы определяются строкой индекса.
Сегментация группирует элементы под повторными индексами, поэтому, например, в нашем случае мы имеем сегментированные ids, [0, 0, 1, 2, 2] применяемые к тензору tens1, что означает, что первый и второй массивы будут преобразованы после операции сегментации (в нашем случае суммирования) и получим новый массив, который выглядит (2, 8, 1, 0) = (2+0, 5+3, 3-2, -5+5). Третий элемент в тензоре tens1 нетронутый, потому что он не сгруппирован ни в один повторный индекс, а последние два массива суммируются так же, как и в случае первой группы.

```
import tensorflow as tf import numpy as np
def convert(v, t=tf.float32):

return tf.convert_to_tensor(v, dtype=t) seg_ids = tf.constant([0, 0, 1, 2, 2])
tens1 = convert(np.array([(2, 5, 3, -5), (0, 3, -2, 5), (4, 3, 5, 3), (6, 1, 4, 0), (6, 1, 4,
0)]), tf.int32)

tens2 = convert(np.array([1, 2, 3, 4, 5]), tf.int32) seg_sum = tf.segment_sum(tens1, seg_ids)
seg_sum_1 = tf.segment_sum(tens2, seg_ids) with tf.Session() as session:
print("Segmentation sum tens1: ", session.run(seg_sum))

print("Segmentation sum tens2: ", session.run(seg_sum_1))
```

Вывод:
Segmentation sum tens1: [[2 8 1 0]
[4 3 5 3]
[12 2 8 0]]
Segmentation sum tens2: [3 3 9]

TensorFlow содержит также такие методы, как:
- argmin, которая возвращает индекс с минимальным значением по осям входного тензора,
- argmax, которая возвращает индекс с максимальным значением по осям входного тензора,

• setdiff, который вычисляет разницу между двумя списками чисел или строк.
Ниже мы приводим несколько примеров использования этих функций:
import numpy as np import tensorflow as tf
def convert(v, t=tf.float32):

return tf.convert_to_tensor(v, dtype=t) x = convert(np.array([
[2, 2, 1, 3],

[4, 5, 6, -1],

[0, 1, 1, -2],

[6, 2, 3, 0]

]))

y = convert(np.array([1, 2, 5, 3, 7]))

z = convert(np.array([1, 0, 4, 6, 2])) arg_min = tf.argmin(x, 1)
arg_max = tf.argmax(x, 1) unique = tf.unique(y)
diff = tf.setdiff1d(y, z)

with tf.Session() as session:

print("Argmin = ", session.run(arg_min)) print("Argmax = ", session.run(arg_max)) print("Unique_values = ", session.run(unique)[0]) print("Unique_idx = ", session.run(unique)[1]) print("Setdiff_values = ", session.run(diff)[0]) print("Setdiff_idx = ", session.run(diff)[1]) print(session.run(diff)[1])
Вывод:

Argmin = [2 3 3 3]

Argmax = [3 2 1 0]

Unique_values = [1. 2. 5. 3. 7.]

Unique_idx = [0 1 2 3 4]

Setdiff_values = [5. 3. 7.]

Setdiff_idx = [2 3 4]

[2 3 4]

18.8. Пример нейронной сети в TensorFlow

Рассмотрим пример применения TensorFlow для создания простой трехслойной нейронной сети. В этом примере мы будем использовать набор данных MNIST (и связанный с ним загрузчик), который предоставляет пакет TensorFlow. Этот набор данных MNIST представляет собой набор изображений в оттенках серого размером 28 × 28 пикселей, которые представляют собой рукописные цифры. Он имеет 55 000 учебных рядов, 10 000 строк тестирования и 5000 строк проверки.

Мы можем загрузить данные, запустив:

```
import tensorflow as tf import numpy as np
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

One_hot=True - аргумент указывает, что вместо меток, связанных с каждым проецирование изображения, сама цифра, то есть «4», то есть вектор с
«один горячий» узел и все остальные узлы равно нулю, то есть [0, 0, 0, 0 , 1, 0, 0, 0, 0, 0]. Это позволяет нам легко подавать его в выходной слой нашей нейронной сети.
Затем мы можем настроить переменные-плейсхолдеры для данных обучения (и некоторые параметры обучения):

```
# Python optimisation variables

learning_rate = 0.5

epochs = 10

batch_size = 100

# declare the training data placeholders # input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])

# now declare the output data placeholder - 10 digits y = tf.placeholder(tf.float32, [None, 10])
```

Обратите внимание, что входной уровень X представляет содержит 784 узла, соответствующих 28 x 28 (= 784) пикселам, а выходной уровень Y- 10 узлов, соответствующих 10 возможных разрядов. Опять же, размер X равен (? X 784), где ? обозначает еще не заданное количество выборок, которые нужно ввести - это функция переменной-плейсхолдера.
Теперь нам нужно настроить переменные веса и смещения для трехслойной нейронной сети. Всегда должно быть L-1 количество тензоров веса / смещения, где L - количество слоев. Поэтому в этом случае нам нужно настроить два тензора для каждого:

```
# now declare the weights connecting the input to the hidden layer

W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1') b1 = tf.Variable(tf.random_normal([300]), name='b1')
# and the weights connecting the hidden layer to the output layer

W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')

b2 = tf.Variable(tf.random_normal([10]), name='b2')
```

В этом коде мы объявляем некоторые переменные для W1 и b1, веса и смещения для связей между входным и скрытым слоями. Эта нейронная сеть будет иметь 300 узлов в скрытом слое, поэтому размер весового тензора W1 равен [784, 300]. Мы инициализируем значения весов, используя случайное нормальное распределение со средним значением равным нулю и стандартным отклонением 0,03. TensorFlow имеет реплицированную версию случайной нормальной функции numpy, которая позволяет вам создать матрицу заданного размера, заполненную случайными выборками, полученными из данного распределения. Аналогично, мы создаем переменные W2 и b2 для подключения скрытого слоя к выходному уровню нейронной сети.
Затем мы должны настроить входы узлов и функции активации узлов скрытого слоя:

```
# calculate the output of the hidden layer hidden_out = tf.add(tf.matmul(x, W1), b1) hidden_out =
```

tf.nn.relu(hidden_out)
В первой строке мы выполняем стандартное матричное умножение весов W1 на входной вектор X и добавляем смещение b1. Матричное умножение выполняется с использованием операции tf.matmul. Затем мы завершаем операцию hidden_out, применяя функцию активации relu к произведению матрицы весов W1 и входа X плюс смещение.
Теперь давайте настроим выходной уровень, y_:

# now calculate the hidden layer output - in this case, let's use a softmax activated # output layer
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))

Снова мы выполняем умножение веса с выходом из скрытого слоя (hidden_out) и добавляем смещение b2 . В этом случае мы будем использовать активацию softmax для выходного уровня.
Мы также должны включить функцию затрат или. Здесь мы будем использовать функцию кросс-энтропии. Мы можем реализовать эту функцию кросс-энтропии в TensorFlow со следующим кодом:

# now let's define the cost function which we are going to train the model on y_clipped = tf.clip_by_value(y_, 1e-10, 0.9999999)
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(y_clipped) + (1 - y) * tf.log(1 - y_clipped), axis=1))

Первая строка - операция, преобразующая выход y_clipped версию, ограниченную между 1e-10 и 0.999999. Вторая строка - расчет кросс-энтропии. Чтобы выполнить этот расчет, сначала мы используем функцию tf.reduce_sum, которая берет сумму заданной оси
тензора.
Давайте настроим оптимизатор в TensorFlow:

add an optimiser

optimizer= tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cross_entropy)

Здесь мы просто используем оптимизатор градиентного спуска, предоставляемый с TensorFlow. Мы инициализируем его с помощью скорости обучения, а затем указываем, что мы хотим сделать, то есть свести к минимуму транзакционную операцию кросс-энтропии, которую мы создали. Затем эта функция выполнит градиентный спуск.
Далее настроим операцию инициализации переменных и операцию для измерения точности наших прогнозов:

# finally setup the initialisation operator init_op = tf.global_variables_initializer() # define an accuracy assessment operation
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1)) accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)) # add a summary to store the accuracy tf.summary.scalar('accuracy', accuracy)
Операция предсказания correct_prediction использует функцию tf.equal TensorFlow, которая возвращает True или False в зависимости от того, равны ли его аргументы. Функция tf.argmax совпадает с функцией numpy argmax, которая возвращает индекс максимального значения в векторе или тензоре. Поэтому операция correct_prediction возвращает тензор размера (m x 1) True и False значения, определяющие, правильно ли предсказала цифру нейронная сеть. Затем мы хотим вычислить среднюю точность из этого тензора
- сначала мы должны отличить тип операции correct_prediction от булева до плавающего TensorFlow, чтобы выполнить операцию reduce_mean. Как только мы это сделаем, теперь у нас есть функция точности, которая готова оценить производительность нашей нейронной

сети.
Теперь у нас есть все необходимое для настройки процесса обучения нашей нейронной сети. Запускаем процесс обучения нейронной сети.

merged = tf.summary.merge_all() writer = tf.summary.FileWriter('C:\\D') # start the session

with tf.Session() as sess: sess.run(init_op)
total_batch = int(len(mnist.train.labels) / batch_size) for epoch in range(epochs):
avg_cost = 0

for i in range(total_batch):

batch_x, batch_y = mnist.train.next_batch(batch_size=batch_size)

_, c = sess.run([optimiser, cross_entropy], feed_dict={x: batch_x,
y: batch_y})

avg_cost += c / total_batch
print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost)) summary = sess.run(merged, feed_dict={x: mnist.test.images, y:
mnist.test.labels})

writer.add_summary(summary, epoch) print("\nTraining complete!") writer.add_graph(sess.graph)
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))

Запуск этой программы дает следующий результат: Epoch: 1 cost = 0.586
Epoch: 2 cost = 0.213

Epoch: 3 cost = 0.150

Epoch: 4 cost = 0.113

Epoch: 5 cost = 0.094

Epoch: 6 cost = 0.073

Epoch: 7 cost = 0.058

Epoch: 8 cost = 0.045

Epoch: 9 cost = 0.036

Epoch: 10 cost = 0.027 Training complete!
0.9787

Мы получаем примерно 98% точности на тестовом наборе что довольно неплохо для данной задачи. Мы могли бы сделать несколько вещей, чтобы улучшить модель, например, регуляризацию, но здесь нас просто интересует исследование основ TensorFlow. Вы также можете использовать визуализацию TensorBoard, чтобы посмотреть на повышение точности работы нейронной сети при обучении

7. Оценочные материалы промежуточной аттестации

Очная форма обучения, Третий семестр, Зачет
Контролируемые ИДК: ПК-П12.1 ПК-П15.1 ПК-П1.1 ПК-П12.2 ПК-П15.2 ПК-П1.2 ПК-П1.3 ПК-П12.3 ПК-П15.3
Вопросы/Задания:

1. 3. КЛАССИФИКАЦИЯ НЕЙРОННЫХ СЕТЕЙ

Классификация нейронных сетей по характеру обучения делит их на:
- нейронные сети, использующие обучение с учителем;
- нейронные сети, использующие обучение без учителя.

Классификация нейронных сетей по типу настройки весов делит их на:
- сети с фиксированными связями – весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи;
- сети с динамическими связями – для них в процессе обучения происходит настройка синаптических весов.

Классификация нейронных сетей по типу входной информации делит их на:
- аналоговые – входная информация представлена в форме действительных чисел;
- двоичные – вся входная информация в таких сетях представляется в виде

Заочная форма обучения, Четвертый семестр, Зачет
Контролируемые ИДК: ПК-П12.1 ПК-П15.1 ПК-П1.1 ПК-П12.2 ПК-П15.2 ПК-П1.2 ПК-П1.3 ПК-П12.3 ПК-П15.3
Вопросы/Задания:

1. Классификация нейронных сетей

Классификация нейронных сетей по характеру обучения делит их на:
- нейронные сети, использующие обучение с учителем;
- нейронные сети, использующие обучение без учителя.

Классификация нейронных сетей по типу настройки весов делит их на:
- сети с фиксированными связями – весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи;
- сети с динамическими связями – для них в процессе обучения происходит настройка синаптических весов.

Классификация нейронных сетей по типу входной информации делит их на:
- аналоговые – входная информация представлена в форме действительных чисел;
- двоичные – вся входная информация в таких сетях представляется в виде нулей и единиц.

2. Реализуйте работающий пример нейронной сети в TensorFlow

Рассмотрим пример применения TensorFlow для создания простой трехслойной нейронной сети. В этом примере мы будем использовать набор данных MNIST (и связанный с ним загрузчик), который предоставляет пакет TensorFlow. Этот набор данных MNIST представляет собой набор изображений в оттенках серого размером 28 × 28 пикселей, которые представляют собой рукописные цифры. Он имеет 55 000 учебных рядов, 10 000 строк тестирования и 5000 строк проверки.

Мы можем загрузить данные, запустив:

```
import tensorflow as tf import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
```

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

One_hot=True - аргумент указывает, что вместо меток, связанных с каждым проецирование изображения, сама цифра, то есть «4», то есть вектор с
«один горячий» узел и все остальные узлы равно нулю, то есть [0, 0, 0, 0 , 1, 0, 0, 0, 0, 0]. Это позволяет нам легко подавать его в выходной слой нашей нейронной сети.
Затем мы можем настроить переменные-плейсхолдеры для данных обучения (и некоторые параметры обучения):

Python optimisation variables

learning_rate = 0.5

epochs = 10

batch_size = 100

# declare the training data placeholders # input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])

now declare the output data placeholder - 10 digits y = tf.placeholder(tf.float32, [None, 10])
Обратите внимание, что входной уровень X представляет содержит 784 узла, соответствующих 28 x 28 (= 784) пикселам, а выходной уровень Y- 10 узлов, соответствующих 10 возможных разрядов. Опять же, размер X равен (? X 784), где ? обозначает еще не заданное количество выборок, которые нужно ввести - это функция переменной-плейсхолдера.
Теперь нам нужно настроить переменные веса и смещения для трехслойной нейронной сети. Всегда должно быть L-1 количество тензоров веса / смещения, где L - количество слоев. Поэтому в этом случае нам нужно настроить два тензора для каждого:

now declare the weights connecting the input to the hidden layer

W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1') b1 = tf.Variable(tf.random_normal([300]), name='b1')
# and the weights connecting the hidden layer to the output layer

W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')

b2 = tf.Variable(tf.random_normal([10]), name='b2')

В этом коде мы объявляем некоторые переменные для W1 и b1, веса и смещения для связей между входным и скрытым слоями. Эта нейронная сеть будет иметь 300 узлов в скрытом слое, поэтому размер весового тензора W1 равен [784, 300]. Мы инициализируем значения весов, используя случайное нормальное распределение со средним значением равным нулю и стандартным отклонением 0,03. TensorFlow имеет реплицированную версию случайной нормальной функции numpy, которая позволяет вам создать матрицу заданного размера, заполненную случайными выборками, полученными из данного распределения. Аналогично, мы создаем переменные W2 и b2 для подключения скрытого слоя к выходному уровню нейронной сети.
Затем мы должны настроить входы узлов и функции активации узлов скрытого слоя:

calculate the output of the hidden layer hidden_out = tf.add(tf.matmul(x, W1), b1) hidden_out = tf.nn.relu(hidden_out)
В первой строке мы выполняем стандартное матричное умножение весов W1 на входной

вектор X и добавляем смещение b1. Матричное умножение выполняется с использованием операции tf.matmul. Затем мы завершаем операцию hidden_out, применяя функцию активации relu к произведению матрицы весов W1 и входа X плюс смещение.
Теперь давайте настроим выходной уровень, y_:

# now calculate the hidden layer output - in this case, let's use a softmax activated # output layer
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))

Снова мы выполняем умножение веса с выходом из скрытого слоя (hidden_out) и добавляем смещение b2 . В этом случае мы будем использовать активацию softmax для выходного уровня.
Мы также должны включить функцию затрат или. Здесь мы будем использовать функцию кросс-энтропии. Мы можем реализовать эту функцию кросс-энтропии в TensorFlow со следующим кодом:

# now let's define the cost function which we are going to train the model on y_clipped = tf.clip_by_value(y_, 1e-10, 0.9999999)
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(y_clipped) + (1 - y) * tf.log(1 - y_clipped), axis=1))

Первая строка - операция, преобразующая выход y_clipped версию, ограниченную между 1e-10 и 0.999999. Вторая строка - расчет кросс-энтропии. Чтобы выполнить этот расчет, сначала мы используем функцию tf.reduce_sum, которая берет сумму заданной оси
тензора.
Давайте настроим оптимизатор в TensorFlow:

add an optimiser

optimizer= tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cross_entropy)

Здесь мы просто используем оптимизатор градиентного спуска, предоставляемый с TensorFlow. Мы инициализируем его с помощью скорости обучения, а затем указываем, что мы хотим сделать, то есть свести к минимуму транзакционную операцию кросс-энтропии, которую мы создали. Затем эта функция выполнит градиентный спуск.
Далее настроим операцию инициализации переменных и операцию для измерения точности наших прогнозов:

# finally setup the initialisation operator init_op = tf.global_variables_initializer() # define an accuracy assessment operation
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1)) accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32)) # add a summary to store the accuracy tf.summary.scalar('accuracy', accuracy)
Операция предсказания correct_prediction использует функцию tf.equal TensorFlow, которая возвращает True или False в зависимости от того, равны ли его аргументы. Функция tf.argmax совпадает с функцией numpy argmax, которая возвращает индекс максимального значения в векторе или тензоре. Поэтому операция correct_prediction возвращает тензор размера (m x 1) True и False значения, определяющие, правильно ли предсказала цифру нейронная сеть. Затем мы хотим вычислить среднюю точность из этого тензора
- сначала мы должны отличить тип операции correct_prediction от булева до плавающего TensorFlow, чтобы выполнить операцию reduce_mean. Как только мы это сделаем, теперь у нас есть функция точности, которая готова оценить производительность нашей нейронной сети.
Теперь у нас есть все необходимое для настройки процесса обучения нашей нейронной сети.

Запускаем процесс обучения нейронной сети.

merged = tf.summary.merge_all() writer = tf.summary.FileWriter('C:\\D') # start the session

with tf.Session() as sess: sess.run(init_op)
total_batch = int(len(mnist.train.labels) / batch_size) for epoch in range(epochs):
avg_cost = 0

for i in range(total_batch):

batch_x, batch_y = mnist.train.next_batch(batch_size=batch_size)

_, c = sess.run([optimiser, cross_entropy], feed_dict={x: batch_x,
y: batch_y})

avg_cost += c / total_batch
print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost)) summary = sess.run(merged, feed_dict={x: mnist.test.images, y:
mnist.test.labels})

writer.add_summary(summary, epoch) print("\nTraining complete!") writer.add_graph(sess.graph)
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))

Запуск этой программы дает следующий результат: Epoch: 1 cost = 0.586
Epoch: 2 cost = 0.213

Epoch: 3 cost = 0.150

Epoch: 4 cost = 0.113

Epoch: 5 cost = 0.094

Epoch: 6 cost = 0.073

Epoch: 7 cost = 0.058

Epoch: 8 cost = 0.045

Epoch: 9 cost = 0.036

Epoch: 10 cost = 0.027 Training complete!
0.9787

Мы получаем примерно 98% точности на тестовом наборе что довольно неплохо для данной задачи. Мы могли бы сделать несколько вещей, чтобы улучшить модель, например, регуляризацию, но здесь нас просто интересует исследование основ TensorFlow. Вы также можете использовать визуализацию TensorBoard, чтобы посмотреть на повышение точности работы нейронной сети при обучении

Контролируемые ИДК: ПК-П12.1 ПК-П15.1 ПК-П1.1 ПК-П12.2 ПК-П15.2 ПК-П1.2 ПК-П1.3 ПК-П12.3 ПК-П15.3

Вопросы/Задания:

1. Где найти Задание-инструкцию по разработке собственного интеллектуального облачного Эйдос-приложения для контрольной работы?

Задание-инструкция по разработке собственного интеллектуального облачного Эйдос-приложения
DOI: 10.13140/RG.2.2.27946.44488, License: CC BY-SA 4.0
http://lc.kubagro.ru/aidos/How_to_make_your_own_cloud_Eidos-application.pdf

№ Содержание этапа работ

1 Читаем: http://lc.kubagro.ru/aidos/Presentation_Aidos-online.pdf, http://lc.kubagro.ru/Presentation_LutsenkoEV.pdf
Скачиваем здесь: http://lc.kubagro.ru/aidos/_Aidos-X.htm и устанавливаем на своем компьютере систему «Эйдос».
Ссылка на краткую инструкцию по установке системы «Эйдос» на университетские компьютеры и по первым занятиям по системе, особенно если на занятии отсутствует проф.Е.В.Луценко:
http://lc.kubagro.ru/aidos/Installation_instructions_and_the_first_lessons_on_the_Eidos_system.pdf

2 Запускаем систему «Эйдос», в режиме 1.3, устанавливаем и осваиваем встроенную в полную инсталляцию базовую лабораторную работу: ЛР-3.03. Эта лабораторная работа очень подробно рассматривается во многих видео- занятиях.
Затем изучаем приложения по интеллектуальному анализу текстов (ЛР-3.02) спектральному АСК-анализу изображений (облачное Эйдос-приложение №277):

– видео-занятие по АСК-анализу текстов и спектральному АСК-анализу изображений:https://disk.yandex.ru/i/WoIb6aF4bTuA0Q;

– прямая ссылка на скачивание учебного архива изображений из облачного Эйдос-приложения №277: https://lc.kubagro.ru/Source_data_applications/Applications-000277/Artists_as_classes.rar

По желанию изучаем облачные Эйдос-приложения, отдавая приоритет более новым, т.к. они лучше отражают возможности текущей версии системы «Эйдос» и описаны по более совершенному шаблону описания.
Ссылки на видео-занятия и работы проф.Е.В.Луценко:

– в Пермском национальном университете: https://bigbluebutton.pstu.ru/b/w3y-2ir-ukd-bqn (2021), https://bigbluebutton.pstu.ru/b/3kc-n8a-gon-tjz (2022)

– всегда актуальный каталог видеозанятий проф.Е.В.Луценко по АСК-анализу и системе Эйдос в Кубанском государственном университете и Кубанском государственном аграрном университете: https://lc.kubagro.ru/Video_lessons_by_Prof.E.V.Lutsenko/Catalog.php

– ссылки на работы проф.Е.В.Луценко по различной тематике в открытом доступе: http://lc.kubagro.ru/aidos/index.htm и http://lc.kubagro.ru/aidos/_Aidos-X.htm работы по АСК-анализу текстов: http://lc.kubagro.ru/aidos/Works_on_ASK-analysis_of_texts.htm;
работы по АСК-анализу изображений: http://lc.kubagro.ru/aidos/Works_on_ASK-analysis_of_images.htm; работы по сценарному АСК-анализу: http://lc.kubagro.ru/aidos/Works_on_Scenario_ASC-analysis.htm; страница в Ресечгейт: https://www.researchgate.net/profile/Eugene-Lutsenko

3 По ссылке: https://www.researchgate.net/profile/Eugene_Lutsenko/publications изучаем публикации проф.Е.В.Луценко с описанием приложений системы «Эйдос».

4 Ищем тему и исходные данные для собственного интеллектуального облачного Эйдос-приложения:

- тема и содержание работы не должны быть очень сходными с наименованиями и содержанием уже имеющихся в Эйдос-облаке интеллектуальных приложений: http://lc.kubagro.ru/Source_data_applications/WebAppls.html; (это допускается только если качество решения задачи и качество ее описания значительно выше, чем в более ранней аналогичной по тематике ранее);

- если в строке адреса заменить расширение в ссылке на файл readme.pdf на readme.docx, то как правило этот файл скачается (если такой файл есть в облаке);
- исходные данные рекомендуются искать на сайтах: Kaggle и UCI, а также в поисковых системах по запросу: «Наборы данных для машинного обучения» http://archive.ics.uci.edu/ml/datasets
https://www.kaggle.com/datasets https://www.kaggle.com/competitions (приоритет у активных тем) https://www.kaggle.com/kernels
а также по ссылкам на странице: http://lc.kubagro.ru/aidos/p14.htm (ниже таблицы).
Можно использовать также любые другие исходные данные, не противоречащие общепринятым в России морально-этическим нормам и действующему законодательству Российской Федерации.
Ссылки на лучшие (по мнению автора) бесплатные онлайн CSV=>XLS (XLSX) конвертеры:
https://online-converting.ru/documents/csv-to-xls/ (конвертирует CSV-файлы больше 100 Мб) https://convertio.co/ru/csv-xls/
https://onlineconvertfree.com/ru/convert-format/csv-to-xls/
https://document.online-convert.com/ru/convert/csv-to-excel
SCV-стандарт не устоялся, встречается много разных специфических особенностей в CSV-файлах, поэтому иногда лучше подходит один конвертер, а иногда другой. Файл исходных данных: Inp_data.xls, Inp_data.xlsx должен быть меньше 10 Мб, т.к. файлы большего размера автоматически удаляются с ftp-сервера системы «Эйдос». Поэтому важно знать и учитывать, что один и тот же файл в XLSX-стандарте обычно примерно в два раза меньше по размеру, чем в XLS.
Но лучше брать еще меньший объем данных (не мегабайты, а сотни или даже десятки килобайт), тогда длительность расчетов будет более приемлемой.
5 Показываем проф.Е.В.Луценко на занятии или присылаем ссылку на их источник исходных данных и сами эти данные для приложения в виде Excel- или CSV-файла в стандарте программного интерфейса (API) 2.3.2.2 системы
«Эйдос» и примерную тему на эл.почту проф.Е.В.Луценко: prof.lutsenko@gmail.com для утверждения. Утверждение возможно только в том, случае, если модель получается достаточно достоверная или хотя бы разумная. После утверждения темы можно выполнять следующие пункты.
6 Описываем созданное Эйдос-приложение, взяв за образец (т.е. в качестве шаблона описания) вордовский файл одной из статей:
1. Луценко Е.В. Автоматизированный системно-когнитивный анализ силы и направления влияния морфологических свойств помидоров на количественные, качественные и финансово-экономические результаты их выращивания и степень детерминированности этих результатов в условиях неотапливаемых теплиц Юга России / Е.В. Луценко, Р.А. Гиш, Е.К. Печурина, С.С. Цыгикало // Политематический сетевой электронный научный журнал Кубанского государственного аграрного университета (Научный журнал КубГАУ) [Электронный ресурс]. – Краснодар: КубГАУ, 2019. – №06(150). С. 79 – 129. – IDA [article ID]: 1501906015. – Режим доступа:
http://ej.kubagro.ru/get.asp?id=7763&t=2, 3,188 у.п.л.

2. Тематические подборки публикаций по применению АСК-анализа и системы «Эйдос» в различных предметных областях: http://lc.kubagro.ru/aidos/_Aidos-X.htm#_Toc99666361
3. Шаблон описания научного исследования с применением АСК-анализа и системы Эйдос (стандарт IMRAD): Детальный пример описания научного исследования в файле WORD: https://www.researchgate.net/publication/362211691. Для скачивания файла надо кликнуть по синей кнопке: «Download the pdf». СКАЧАЕТСЯ ВОРДОВСКИЙ ФАЙЛ. ЧТОБЫ ИЗБЕЖАТЬ ТИПИЧНЫХ ОШИБОК, НЕОБХОДИМО ОБЯЗАТЕЛЬНО написать наименование приложения в режиме 1.3. В подразделе 3.2 описания надо вставить таблицу исходных данных (или ее фрагмент), прямую активную (действующую) ссылку на источник данных этой таблицы в Internet. В КАЖДОМ ПОДРАЗДЕЛЕ шаблона описания, начиная с описания результатов и далее скриншоты экранных форм соответствующих режимов системы «Эйдос» (Alt+PrScreen) и формируемые ей таблицы, а также текст их интерпретации или пояснения.

4. КАК ПРИНИМАЕТСЯ РАБОТА: проф.Е.В.Луценко вводит исходные данные из файла Inp_data.xls(x), присланного учащимся, в систему «Эйдос» с параметрами (_2_3_2_2.arx), указанными в описании readme.doc(x), создает модели в соответствии с параметрами, приведенными в описании и сравнивает выходные формы, получающиеся в модели с выходными формами, приведенными в описании. Если они совпадают – работа принимается, а иначе отклоняется на доработку. В частности текст с описанием результатов оценки достоверности моделей в режиме 3.4 должен соответствовать экранным формам этого режима. Если получающиеся экранные формы нечитабельны, то используя параметры настройки изображений сделать их читабельными. Аккуратно отформатировать описание: рисунки на одной странице должны быть одной ширины.
ВАЖНО!!! Внимательно смотрите, чтобы в итоговом описании, если оно посвящено, например, процессорам или видеокартам, ничего не осталось про геном, помидоры, морфологические и биохимические свойства, урожайность, жирность и т.п.
7 Показываем проф.Е.В.Луценко на занятии или присылаем исходные данные для приложения в виде Excel-файла в стандарте программного интерфейса (API) 2.3.2.2 системы «Эйдос» и описание приложения (файлы:
Inp_data.xls(x), readme.doc(x), c:\Aidos-X_2_3_2_2.arx) на эл.почту проф.Е.В.Луценко: prof.lutsenko@gmail.com для принятия решения и, в случае если оно положительное, то и для размещения созданного приложения и его
описания в Эйдос-облаке, и только описания в ResearchGate и в РИНЦ. Главных критерия приема работы два: 1) созданные мной на основе ваших данных модели совпадают с вашими; 2) ваше описание соответствует вашим данным и созданным на основе них вашим моделям.
8 Само размещение Эйдос-приложения в облаке для учащихся осуществляет лично проф.Е.В.Луценко. Размещение описания приложения в ResearchGate и в РИНЦ можно осуществлять только после их просмотра проф.Е.В.Луценко и одобрения этого им. Размещение описания приложения в ResearchGate и в РИНЦ осуществляет учащийся или соавтор. Для этого он должен зарегистрироваться или уже быть зарегистрированным в ResearchGate: https://www.researchgate.net/, а также в https://elibrary.ru/ и системе в SCIENCE INDEX, получить SPIN-код и заключить с РИНЦ договор на физическое лицо на размещение непериодических изданий в РИНЦ:
https://elibrary.ru/projects/contracts/publisher/messages/messages.asp? Подробнее см. здесь: http://lc.kubagro.ru/ResearchGate.doc.
9 Оценка знаний, умений и навыков, полученных учащимися при освоении АСК-анализа и системы «Эйдос»

Оценка Разработка и размещение Эйдос-приложения в: Стоимость сертификата ВЦСКИ "Эйдос", подтверждающего учебное достижение по освоению АСК-анализа и системы "Эйдос" (в рублях по курсу
USD, ЦБ РФ) (по желанию*)

Ссылки на образцы сертификатов
Эйдос-облако
(исходные данные и описание по шаблону) ResearchGate
(только описание по шаблону) РИНЦ (только
описание по шаблону)
Отлично Да Да Да 100 Шаблон по выбору получателя
Хорошо Да Да Нет 50 Шаблон по выбору получателя
Удовлетворительно Да Нет Нет 25 Шаблон по выбору получателя
По результатам аттестации Нет Нет Нет *** ***
* Это предложение не касается учащихся тех вузов, в которых работает автор
10 Если учащийся не зарегистрировался в ResearchGate (для этого необходим корпоративный адрес электронной почты от НИИ или Университета) и в РИНЦ, то описания облачных

Эйдос-приложений могут быть размещены в ResearchGate (https://www.researchgate.net/profile/Eugene_Lutsenko) в качестве препринтов с присвоением DOI, а затем будут размещены в РИНЦ (https://elibrary.ru/) в качестве публикаций в открытом архиве, т.е. войдут в список
публикаций учащегося и его портфолио. Но для этого будет необходимо включить проф.Е.В.Луценко в качестве соавтора в описание приложения, т.к. размещать материалы в этих системах могут только их авторы.
11 Литература: https://www.researchgate.net/profile/Eugene_Lutsenko/publications
12 On-line консультации проф.Е.В.Луценко по всем вопросам, связанным с созданием и размещением облачного Эйдос-приложения: https://www.researchgate.net/profile/Eugene_Lutsenko
или по e-mail: prof.lutsenko@gmail.com

Базы данных, необходимые для описания облачного Эйдос-приложения
Class_Sc.dbf Классификационные шкалы
Opis_Sc.dbf Описательные шкалы
Classes.dbf Классификационные шкалы и градации
Attributes.dbf Описательные шкалы и градации
EventsKO.dbf База событий (обучающая или тренировочная выборка)
Базы данных и выходные формы по значимости описательных шкал и градаций и степени детерминированности классификационных шкал и градаций формируются в режимах 3.7.2, 3.7.3, 3.7.4 и 3.7.5 системы Эйдос. В этих же режимах в конце выводится информация об именах и месте расположения выходных баз данных.
Режим 5.12 системы Эйдос преобразует все dbf-файлы в папке текущего приложения в xls-файлы, которые открываются в MS Excel. Текущее приложение находится по пути: ..\Aidos-X\AID_DATA\A0000001\System\.
Вообще после выполнения любого режима системы «Эйдос» формируемые им базы данных будут в начале списка файлов, если в файл-менеджере выбрать сортировку по времени создания.

8. Материально-техническое и учебно-методическое обеспечение дисциплины

8.1. Перечень основной и дополнительной учебной литературы

Основная литература

1. Филиппов Ф. В. Нейросетевые технологии: учебное пособие / Филиппов Ф. В.. - Санкт-Петербург: СПбГУТ им. М.А. Бонч-Бруевича, 2020. - 129 с. - Текст: электронный. // RuSpLAN: [сайт]. - URL: https://e.lanbook.com/img/cover/book/180056.jpg (дата обращения: 21.02.2024). - Режим доступа: по подписке

Дополнительная литература

1. Филиппов Ф. В. Нейросетевые технологии: лабораторный практикум / Филиппов Ф. В.. - Санкт-Петербург: СПбГУТ им. М.А. Бонч-Бруевича, 2021. - 50 с. - Текст: электронный. // RuSpLAN: [сайт]. - URL: https://e.lanbook.com/img/cover/book/279539.jpg (дата обращения: 21.02.2024). - Режим доступа: по подписке

8.2. Профессиональные базы данных и ресурсы «Интернет», к которым обеспечивается доступ обучающихся

Профессиональные базы данных

1. https://kpfu.ru/staff_files/F1493580427/NejronGafGal.pdf - Гафаров Ф.М. Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.

2. http://lc.kubagro.ru/Installation_Eidos.php - Всегда актуальная информация об установочных файлах системы Эйдос

3. http://lc.kubagro.ru/Source_data_applications/WebAppls.html - Актуальный каталог интеллектуальных облачных Эйдос-приложений (датасеты + описания решения в системе Эйдос):

4. http://lc.kubagro.ru/Video_lessons_by_Prof.E.V.Lutsenko/Catalog.php - Всегда актуальный каталог видеозанятий проф.Е.В.Луценко по АСК-анализу и системе Эйдос

5. https://www.researchgate.net/profile/Eugene-Lutsenko - Страничка проф.Е.В.Луценко в РесечГейт

6. http://www.researchgate.net/publication/340000414 - Луценко Е. В. Интеллектуальные информационные системы : учебник / Е. В. Луценко. – Краснодар : ВЦСКИ «Эйдос», 2021. – 529 с. http://www.researchgate.net/publication/340000414

7. http://www.researchgate.net/publication/365302016 - Луценко Е. В. Методы искусственного интеллекта : учебник // Е. В. Луценко. – Краснодар : ВЦСКИ «Эйдос», 2020. – 520 с., November 2022, DOI: 10.13140/RG.2.2.23807.07847, License CC BY 4.0, http://www.researchgate.net/publication/365302016

Ресурсы «Интернет»

Не используются.

8.3. Программное обеспечение и информационно-справочные системы, используемые при осуществлении образовательного процесса по дисциплине

Перечень программного обеспечения
(обновление производится по мере появления новых версий программы)

Не используется.

Перечень информационно-справочных систем
(обновление выполняется еженедельно)

Не используется.

8.4. Специальные помещения, лаборатории и лабораторное оборудование

9. Методические указания по освоению дисциплины (модуля)

Учебная работа по направлению подготовки осуществляется в форме контактной работы с преподавателем, самостоятельной работы обучающегося, текущей и промежуточной аттестаций, иных формах, предлагаемых университетом. Учебный материал дисциплины структурирован и его изучение производится в тематической последовательности. Содержание методических указаний должно соответствовать требованиям Федерального государственного образовательного стандарта и учебных программ по дисциплине. Самостоятельная работа студентов может быть выполнена с помощью материалов, размещенных на портале поддержки Moodl.

10. Методические рекомендации по освоению дисциплины (модуля)

Гафаров Ф.М. Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.