МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. И.А. БУНИНА»

О.Ю. Андропова, И.И. Васильева, Н.А. Гнездилова

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЯЗЫК ПРОГРАММИРОВАНИЯ Python

Учебное пособие

Елец – 2024

УДК 004.42
ББК 32.973
А 66

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А. Бунина
от 29.02.2024, протокол № 1

Рецензенты
И.Н. Коновалова, кандидат педагогических наук,
доцент кафедры гуманитарных и естественнонаучных дисциплин
Липецкого филиала РАНХиГС;
С.С. Бунеев, кандидат физико-математических наук, доцент кафедры
технологических процессов в машиностроении и агроинженерии
Елецкого государственного университета им. И.А. Бунина

О.Ю. Андропова, И.И. Васильева, Н.А. Гнездилова

А 66 Искусственный интеллект и язык программирования Python: учебное пособие. – Елец: Елецкий государственный университет им. И.А. Бунина, 2024. – 106 с.
ISBN 978-5-00151-413-8

Учебное пособие содержит краткое изложение таких теоретических вопросов, как нейронные сети, методы машинного обучения, задача построения нейронной сети для осуществления прогнозирования и расчета значений процесса в программе MATLAB, общие сведения о языке программирования Python для искусственного интеллекта и машинного обучения, способы реализации нейронных сетей в Pytho, практикум применения искусственного интеллекта на Python.

Пособие будет полезно студентам направлений подготовки 09.03.01 Информатика и вычислительная техника, 01.03.02 Прикладная математика и информатика, а также студентам IT-направлений, в учебные планы которых входит дисциплина «Искусственный интеллект».

УДК 004.42
ББК 32.973

ISBN 978-5-00151-413-8

ВВЕДЕНИЕ

Искусственный интеллект (ИИ) – это научная дисциплина, изучающая создание интеллектуальных агентов, то есть систем, которые могут воспринимать окружающую среду и принимать решения, направленные на достижение поставленных целей.

История ИИ началась в 1950-х годах, когда Джон Маккарти, Марвин Мински и другие ученые из Массачусетского технологического института разработали первые программы искусственного интеллекта. Эти программы были достаточно простыми, но они положили начало развитию новой области науки.

В 1960-х годах ИИ добился значительных успехов в области машинного обучения, экспертных систем и искусственного зрения. В это время были созданы такие известные программы, как шахматный движок Deep Blue, система распознавания образов CAPTCHA и экспертная система MYCIN.

В 1970-х годах ИИ столкнулся с кризисом. Многие ученые пришли к выводу, что создание искусственного интеллекта, равного человеческому, является невозможным. В результате финансирование исследований в области ИИ было сокращено, и эта область науки переживала период застоя.

В 1980-х годах ИИ вновь обрел популярность благодаря развитию новых технологий, таких как нейронные сети и параллельные вычисления. В это время были созданы такие успешные программы, как система распознавания речи Siri и программа машинного перевода Google Translate.

В 1990-х годах ИИ продолжил развиваться, и появились новые направления, такие как обработка естественного языка, робототехника и компьютерное зрение. В это время были созданы такие программы, как поисковая система Google, система виртуального помощника Amazon Alexa и программа автономной езды Waymo.

В настоящее время ИИ является одной из наиболее динамично развивающихся областей науки в Российской Федерации.

Основной целью искусственного интеллекта является создание систем, способных обучаться и адаптироваться к новым ситуациям, а также принимать решения на основе имеющейся информации.

Искусственный интеллект включает в себя такие подобласти, как машинное обучение, нейронные сети, обработка естественного языка и компьютерное зрение. Машинное обучение является одной из ключевых техник, используемых в ИИ, при которой компьютерная программа обучается на основе данных и опыта, чтобы улучшить свою производительность в выполнении задач.

Исследования в области ИИ можно разделить на несколько направлений:

1) Эвристическое направление.

Эвристическое направление исследований в ИИ направлено на создание методов решения интеллектуальных задач, которые не обязательно имитируют человеческий интеллект. В рамках этого направления разрабатываются различные алгоритмы и методы, которые позволяют машинам решать задачи, которые ранее считались исключительно прерогативой человека.

Например, в рамках эвристического направления были разработаны методы машинного обучения, которые позволяют машинам обучаться на данных без явного программирования. Эти методы позволили машинам научиться выполнять множество задач, которые ранее считались сложными или невозможными для автоматизации.

2) Бионическое направление.

Бионическое направление исследований в ИИ направлено на создание интеллектуальных агентов, которые имитируют процессы, протекающие в человеческом мозге. В рамках этого направления разрабатываются нейронные сети, которые являются аналогами нейронных сетей головного мозга человека.

Нейронные сети могут обучаться на данных и выполнять различные задачи, которые ранее считались прерогативой человека. Например, нейронные сети используются для распознавания лиц, голосов, изображений и других объектов.

3) Эволюционное направление.

Эволюционное направление исследований в ИИ направлено на создание интеллектуальных агентов, которые развиваются и совершенствуются со временем. В рамках этого направления используются методы эволюционного моделирования, которые позволяют создавать интеллектуальные агенты, которые способны адаптироваться к меняющимся условиям окружающей среды.

Эволюционное направление является одним из наиболее перспективных направлений исследований в области ИИ. Оно позволяет

создавать интеллектуальные агенты, которые способны решать задачи, которые ранее считались невозможными для автоматизации.

Кроме перечисленных выше направлений, в области ИИ также активно развиваются следующие направления:

1) Робототехника

Робототехника – это область ИИ, которая занимается разработкой и созданием роботов. Роботы – это интеллектуальные агенты, которые способны выполнять различные действия в реальном мире.

В последние годы робототехника переживает бурный рост. Роботы используются во всё большем количестве областей, таких как производство, здравоохранение, сфера услуг и т.д.

2) Обработка естественного языка

Обработка естественного языка – это область ИИ, которая занимается разработкой методов обработки и анализа текстовых данных. Обработка естественного языка используется для различных задач, таких как машинный перевод, распознавание речи, генерация текста и т.д.

В последние годы обработка естественного языка также переживает бурный рост. Обработка естественного языка используется во всё большем количестве приложений.

3) Компьютерное зрение

Компьютерное зрение – это область ИИ, которая занимается разработкой методов обработки и анализа изображений. Компьютерное зрение используется для различных задач, таких как распознавание объектов, идентификация лиц, обработка видео и т.д.

В последние годы компьютерное зрение также переживает бурный рост. Компьютерное зрение используется во всё большем количестве приложений, таких как системы безопасности, системы видеонаблюдения и т.д.

ИИ – это быстро развивающаяся область науки и техники. В ближайшие годы ожидается, что ИИ будет использоваться в ещё более широком диапазоне областей.

Одним из наиболее перспективных направлений развития ИИ является создание автономных систем, которые могут действовать независимо от человека. Автономные системы могут использоваться для различных задач, таких как управление транспортом, выполнение опасных работ и т.д.

Другим перспективным направлением развития ИИ является создание интеллектуальных систем, которые могут взаимодейство-

вать с человеком на естественном языке. Такие системы могут использоваться для различных задач, таких как обучение, консультации и т.д.

Развитие ИИ может иметь как положительные, так и отрицательные последствия. С одной стороны, ИИ может помочь решить многие проблемы, с которыми сталкивается человечество. С другой стороны, ИИ может привести к появлению новых рисков, таких как безработица, киберпреступность и т.д.

Важно ответственно подходить к развитию ИИ и минимизировать возможные негативные последствия.

В Python ИИ реализован несколькими подходами, включая машинное обучение, глубокое обучение и нейронные сети. Машинное обучение – это метод обработки данных, позволяющий программе извлекать знания и делать предсказания на основе имеющихся данных. Глубокое обучение – это подмножество машинного обучения, которое использует нейронные сети со множеством слоев для анализа данных. Нейронные сети – это математическая модель, имитирующая работу нервной системы человека.

Одним из самых популярных фреймворков для реализации ИИ в Python является TensorFlow. TensorFlow – это открытый программный фреймворк для машинного обучения, разработанный компанией Google. TensorFlow предоставляет мощные инструменты и библиотеки для разработки и тренировки различных моделей машинного обучения и нейронных сетей. Он обладает обширным набором функций и может быть использован для создания широкого спектра ИИ-приложений.

Реализация ИИ в Python также может включать использование других библиотек и инструментов, таких как Keras, PyTorch, Scikit-learn и NLTK. Keras – это высокоуровневый API для создания нейронных сетей, который может быть использован с TensorFlow. PyTorch – это фреймворк машинного обучения с открытым исходным кодом, разработанный Facebook. Scikit-learn – это библиотека машинного обучения, которая предоставляет набор инструментов для классификации, регрессии, кластеризации и прогнозирования данных. NLTK (Natural Language Toolkit) – это библиотека для обработки естественного языка, которая предоставляет инструменты для анализа текста и работы с языком.

Благодаря множеству инструментов и библиотек в Python, разработка и тренировка ИИ-систем становится все более доступной и простой. Огромный потенциал ИИ в Python позволяет создавать и улучшать различные приложения и технологии, которые способны делать сложные вычисления, анализировать большие объемы данных и принимать решения на основе них, тем самым решая реальные проблемы и улучшая нашу жизнь.

Применение искусственного интеллекта находит свое применение в многих сферах жизни, в том числе в медицине, финансах, производстве, информационных технологиях и даже в развлекательной индустрии.

Медицина: системы искусственного интеллекта могут анализировать медицинские данные и помогать врачам при диагностике и принятии решений. Искусственный интеллект может обрабатывать большие объемы информации, что позволяет выявлять паттерны и зависимости, недоступные для человека.

Финансы: алгоритмы искусственного интеллекта используются для прогнозирования рыночных тенденций, определения оптимальных инвестиционных стратегий и управления рисками. Это позволяет финансовым организациям принимать обоснованные решения на основе данных и улучшать свою эффективность.

Производство: роботы с искусственным интеллектом могут выполнять рутинные задачи, повышая производительность и качество продукции. Они также могут анализировать данные сенсоров и предсказывать возможные неисправности оборудования, что позволяет предпринять меры по их предотвращению.

Информационные технологии: искусственный интеллект используется в области компьютерного зрения, естественного языка, анализа данных и автоматизации процессов. Это позволяет разрабатывать инновационные приложения и сервисы, которые делают нашу жизнь проще и удобнее.

Развлекательная индустрия: системы искусственного интеллекта используются для создания реалистичной графики и анимации, разработки игровых алгоритмов и принятия решений компьютерными противниками. Они также могут предлагать персонализированные рекомендации и содержание на основе предпочтений пользователя.

Настоящее учебное пособие представляет возможности использования языка программирования Python для искусственного интел-

лекта и машинного обучения и предназначено обучающимся и преподавателям образовательной организации для использования в учебной и профессиональной деятельностях.

1. НЕЙРОННЫЕ СЕТИ

Нейронные сети, искусственно выполненные, являются результатом вдохновения биологией, однако их развитие идет далеко за пределами наших знаний о мозге. Хотя мы используем термины и концепции, заимствованные из биологических принципов, наше знание о мозговом функционировании еще недостаточно, мы не имеем доказательной базы многим тенденциям, происходящим в головном мозге, чтобы принять их к руководству.

В этой связи, поиск алгоритмов, методов, структур для реализации определенных функций нейросетей разработчики ведут иногда без опоры на биологические двойники, что приводит к тому, что создаются сети без потенциально допустимого воспроизводства в реальном мире с предположением о несуществующем или не открытом знании функционирования мозга.

Это говорит о том, что при недостаточной связи с биологической наукой нейронные связи по-прежнему пытаются увязать с работой мозга из-за схожести их действий с человеческим познанием, что является неэффективным.

Рассматривая нервную систему человека, мы отмечаем, что она состоит из множества нейронов, каждый из которых имеет качества схожие с другими органами человеческого тела и, при этом, они обладают уникальными свойствами по приему, обработке, передаче сигналов электрохимического содержания по путям нервной системы, составляющим коммуникацию в системе мозговой деятельности.

Эта сложная сеть нейронов позволяет нам воспринимать, анализировать и реагировать на окружающую среду, а также выполнять сложные когнитивные задачи.

В разработке искусственных нейронных сетей мы стремимся создать модели, которые могут имитировать некоторые аспекты работы мозга. Нейроны в искусственных сетях имитируются с помощью математических функций, а связи между ними моделируются с использованием весов, которые определяют силу и направление передачи сигналов. Это позволяет нейронным сетям обучаться на основе больших объемов данных и принимать решения на основе обнаруженных закономерностей.

Однако, несмотря на значительные достижения в области искусственных нейронных сетей, мы все еще далеки от полного понимания

работы мозга и его возможностей. Мозг – это уникальный орган, который обладает сложной структурой и функциональностью, которую пока что не удалось полностью воссоздать в искусственных системах. Тем не менее, исследования в области нейронауки и искусственного интеллекта продолжаются, и мы надеемся, что с течением времени мы сможем получить более глубокое понимание мозга и использовать его принципы для создания более эффективных и интеллектуальных систем.

Рисунок 1 отображает структурные компоненты двух бионейронов.

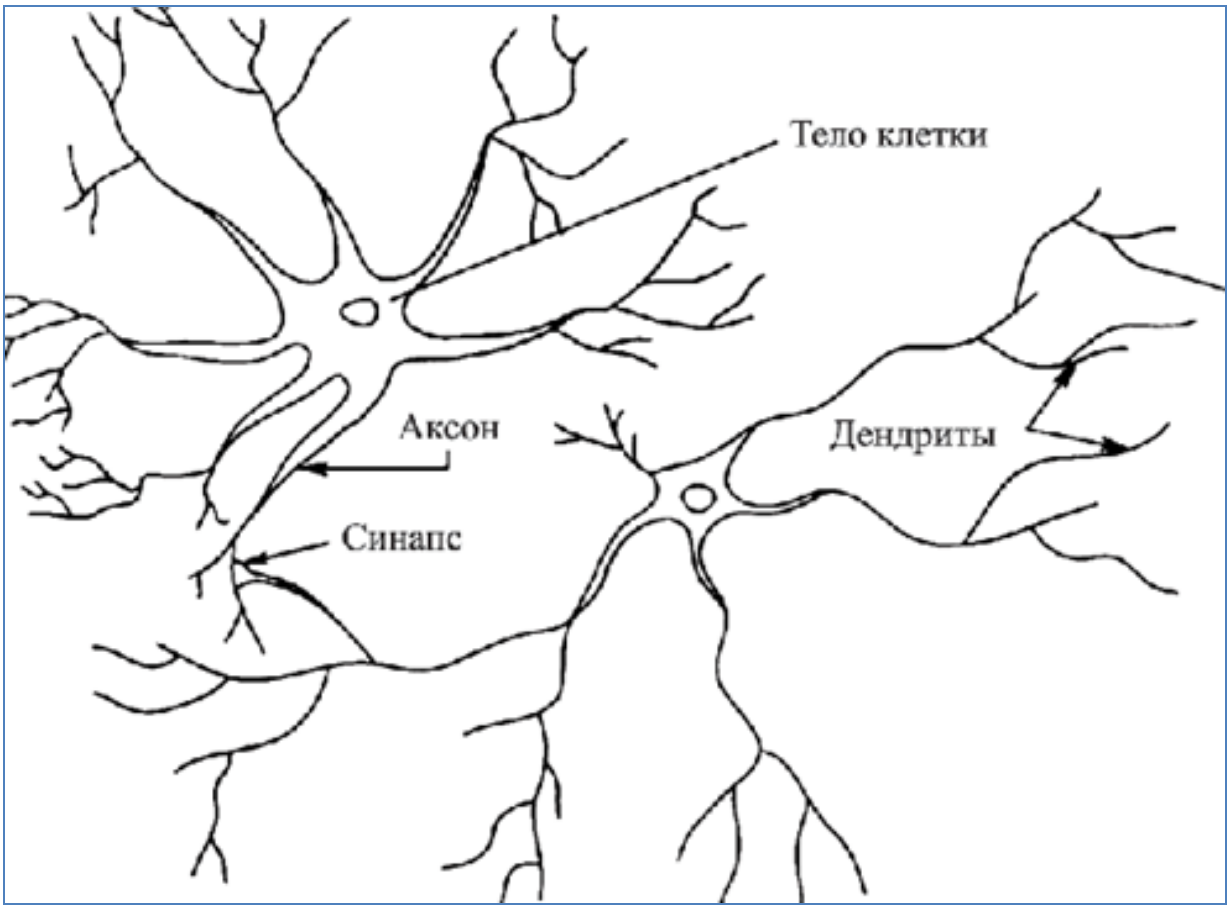

Рисунок 1 – Структурные компоненты двух бионейронов

Нейроны – это основные функциональные единицы нервной системы, которые передают и обрабатывают информацию в мозге и других частях тела. Они состоят из основного тела, от которого отходят многочисленные разветвленные дендриты, получающие сигналы от других нейронов, и длинного аксона, передающего сигналы другим нейронам.

Когда стимулирующий сигнал достигает тела нейрона, он передается в виде электрического импульса, называемого потенциалом действия, который бежит по аксону, вызывая освобождение нейромедиаторов в синаптическую сеть.

Нейромедиаторы – это химические вещества, которые передают сигналы между нейронами, взаимодействуя с рецепторами на дендритах или теле других нейронов.

Дендриты являются основными входными структурами нейрона. Они представляют собой разветвленные отростки, отходящие от тела нейрона. Дендриты покрыты синапсами, которые представляют собой специализированные структуры, позволяющие нейронам обмениваться сигналами друг с другом. Сигнал от одного нейрона к другому нейрону передается через синапсы.

Когда нейромедиаторы связываются с рецепторами, они вызывают изменения в электрической активности принимающего нейрона. Эти изменения могут быть возбуждающими или тормозными. Возбуждающие нейромедиаторы увеличивают вероятность того, что принимающий нейрон сгенерирует потенциал действия, в то время как тормозные нейромедиаторы уменьшают эту вероятность.

Нейроны могут иметь много дендритов, что позволяет им получать сигналы от многих других нейронов. Эти сигналы суммируются в теле нейрона, и если суммарный сигнал превышает определенный порог, нейрон генерирует потенциал действия.

Потенциал действия – это быстрый электрический импульс, который бежит по аксону, вызывая высвобождение нейромедиаторов в синаптическую сеть. Нейромедиаторы затем связываются с рецепторами на дендритах или теле других нейронов, передавая им сигнал.

Нейроны взаимодействуют друг с другом в сложных сетях, образуя нейронные цепи, которые выполняют различные функции в нервной системе. Эти нейронные цепи позволяют организму обрабатывать информацию, учиться и принимать решения.

Рисунок 2 отображает интерпретацию свойств нейрона в биологической системе.

Входные сигналы $x_1, x_2, \dots, x_n$, обозначенные вектором X, направлены на искусственный нейрон и поступают в синапсы бионейрона. Все по отдельности сигналы умножаются на свой соответствующий вес $w_1, w_2, \dots, w_n$, и отправляются на блок $\sum$, складывающий

сумму таких произведений. Соответствующий вес определяет силу синаптической связи в биологической системе.

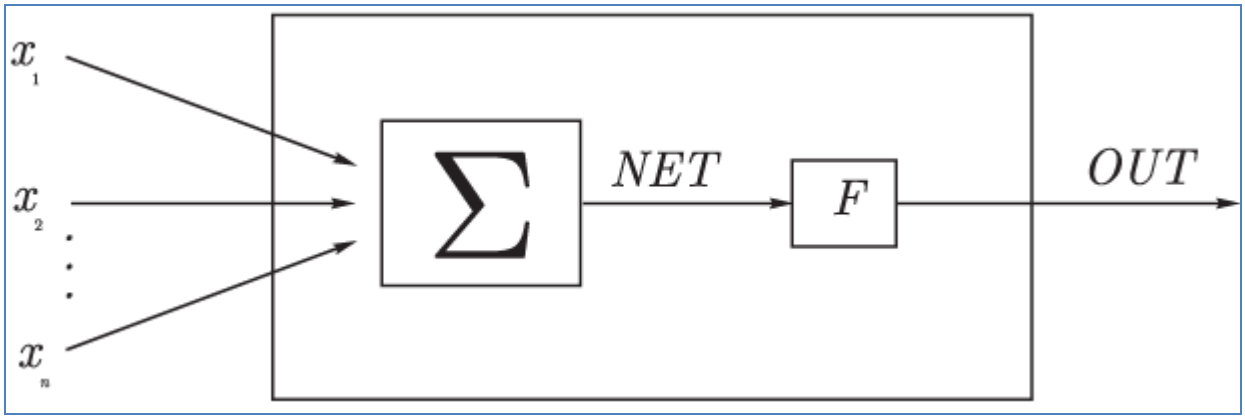

Рисунок 2 – Интерпретация свойств нейрона
в биологической системе

Мы можем представить множество весов в виде вектора W, который обозначает все веса, связанные с определенным биологическим элементом. Эти веса используются для взвешивания входных сигналов, которые затем алгебраически складываются в суммирующем блоке. Таким образом, мы получаем выходной сигнал, который обозначается как NET. NET представляет собой результат алгебраической суммы взвешенных входов. Это позволяет нам оценить степень активации биологического элемента в ответ на входные сигналы. Однако важно отметить, что веса вектора W могут быть изменяемыми и подвержены обучению. В процессе обучения нейронной сети они могут быть настроены таким образом, чтобы достичь оптимальной производительности и точности модели. Также стоит упомянуть, что вектор W может иметь разные размерности в зависимости от количества входных сигналов и требуемой сложности модели. Чем больше входов и сложнее задача, тем больше элементов будет содержать вектор W. Использование векторного представления позволяет нам компактно записывать и оперировать с множеством весов. Это удобно при анализе и оптимизации нейронных сетей, так как мы можем легко изменять и настраивать веса для достижения желаемых результатов:

$$NET = XW.$$

Сигнал NET далее, как правило, преобразуется активационной функцией F и дает выходной нейронный сигнал OUT. Активационная функция может быть обычной линейной функцией

$$OUT = F(NET),$$

Где F – константа, пороговой функцией

$$OUT = \begin{cases} 1, \text{если } NET > T; \\ 0, \text{если } NET < T \end{cases}$$

Где T – некоторая постоянная пороговая величина, или же функция, более точно моделирующая нелинейную передаточную характеристику биологического нейрона и предоставляющей нейронной сети большие возможности.

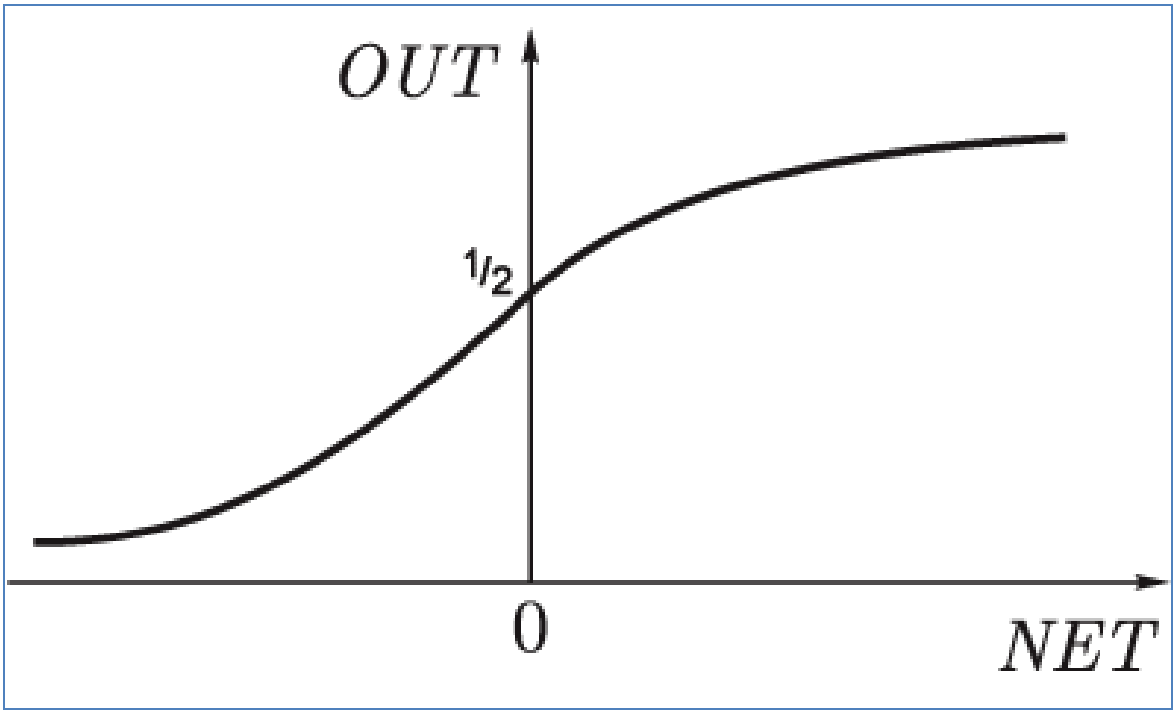

Рисунок 3 – Сжимающая функция
(логистическая или сигмоидальная)

На рисунке 3 блок F принимает сигнал NET и выдает сигнал OUT. Если блок F сужает диапазон изменения величины NET так, что при любых значениях NET значения OUT принадлежат некоторому конечному интервалу, то F называется сжимающей функцией. В ка-

честве сжимающей функции часто используется логистическая или сигмоидальная (S-образная) функция, показанная на рис.3. Эта функция математически выражается как

$$F(x) = 1/(1 + e^{-x}).$$

Таким образом,

$$OUT = \frac{1}{1 + e^{-NET}}.$$

Аналогично электронным системам, активационную функцию искусственного нейрона можно рассматривать как нелинейную характеристику усиления. Коэффициент усиления определяется как отношение приращения OUT к небольшому приращению NET. При определенном уровне возбуждения этот коэффициент выражается наклоном кривой, который изменяется от малых значений при больших отрицательных возбуждениях (кривая практически горизонтальна), достигает максимального значения при нулевом возбуждении, а затем снова уменьшается при увеличении положительного возбуждения.

В 1973 году С. Гроссберг провел исследование, где было обнаружено, что нелинейная характеристика имеет решающее значение для преодоления проблемы шумового насыщения. Каким образом одна и та же сеть способна обрабатывать как слабые, так и сильные сигналы? Для обработки слабых сигналов требуется большое сетевое усиление, чтобы получить выходной сигнал, пригодный для использования. Однако, слишком большие коэффициенты усиления могут привести к насыщению выхода шумами усилителей (случайными флуктуациями), которые присутствуют в любой физически реализованной сети.

Функционирование нейрона основано на использовании широкого диапазона уровня входного сигнала. Центральная область логистической функции, которая имеет высокий коэффициент усиления, позволяет обрабатывать слабые сигналы, а области с падающим усилением на положительном и отрицательном концах подходят для больших возбуждений. Это позволяет избежать насыщения усилительных каскадов и максимально использовать выход. Сильные

входные сигналы также могут быть полезно использованы, но они должны быть в нужном диапазоне для достижения оптимального функционирования нейрона.

$$OUT = \frac{1}{1 + e^{-NET}} = F(NET).$$

Другой широко используемой активационной функцией является гиперболический тангенс. По форме она сходна с логистической функцией и часто используется биологами в качестве математической модели активации нервной клетки. В качестве активационной функции искусственной нейронной сети она записывается следующим образом:

$$OUT = th(x).$$

Подобно логистической функции гиперболический тангенс является S-образной функцией, но он симметричен относительно начала координат, и в точке $NET = 0$ значение выходного сигнала OUT равно нулю (рис. 4). В отличие от логистической функции, гиперболический тангенс принимает значения различных знаков, и это его свойство применяется для целого ряда сетей.

Рассмотренная простая модель искусственного нейрона игнорирует многие свойства своего биологического двойника. Например, она не принимает во внимание задержки во времени, которые воздействуют на динамику системы. Входные сигналы сразу же порождают выходной сигнал. И, что более важно, она не учитывает воздействий функции частотной модуляции или синхронизирующей функции биологического нейрона, которые ряд исследователей считают решающими в нервной деятельности естественного мозга.

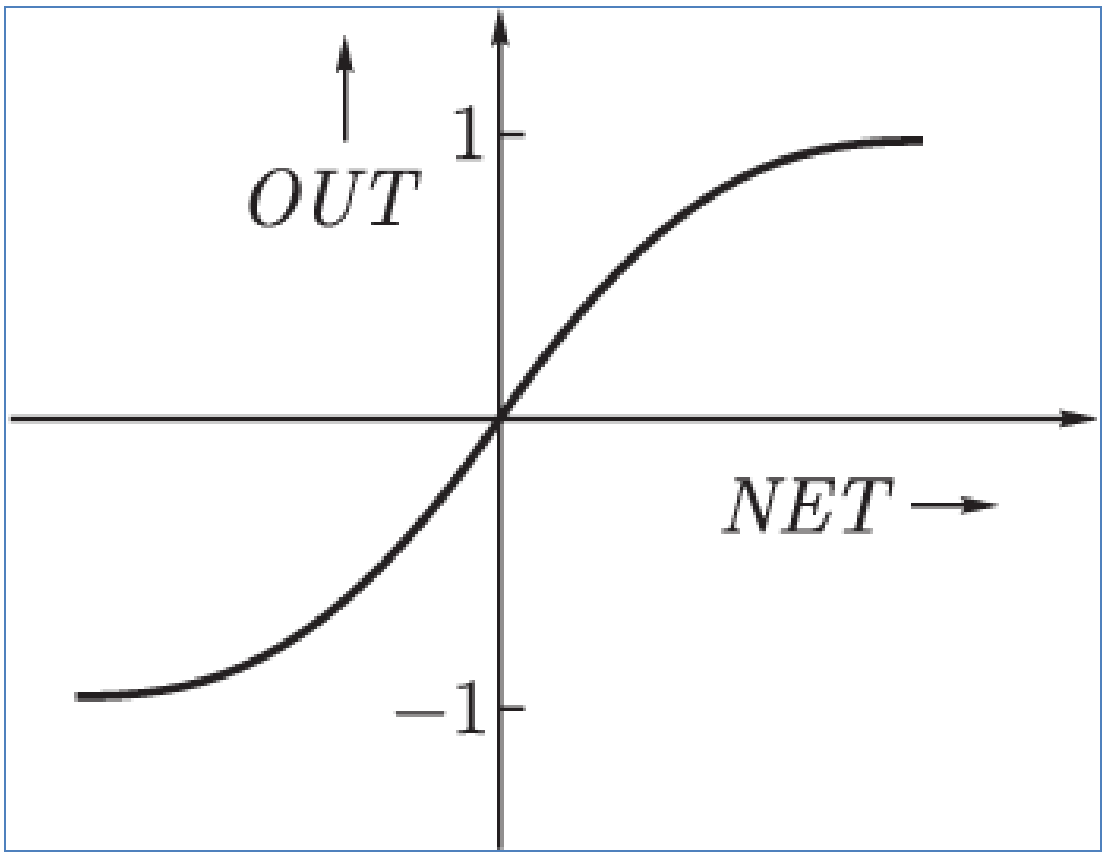

Рисунок 4 – Сжимающая функция (гиперболический тангенс)

Несмотря на эти ограничения, сети, построенные из таких нейронов, обнаруживают свойства, сильно напоминающие биологическую систему. Только время и исследования смогут ответить на вопрос, являются ли подобные совпадения случайными или же они есть следствие того, что в модели верно схвачены важнейшие черты биологического нейрона.

Однослойные искусственные нейронные сети

Однослойные искусственные нейронные сети являются одним из наиболее простых и распространенных типов нейронных сетей. Они состоят из нейронов, которые объединены в один слой, где каждый нейрон соединен с каждым нейроном следующего слоя. Однослойные сети обладают способностью обучаться и обрабатывать информацию, причем их функциональность может быть применена во многих областях, включая распознавание образов, анализ данных, прогнозирование и решение задач классификации. Однако, однослойные сети имеют свои ограничения, такие как невозможность решения сложных проблем, требующих высокой степени абстракции. В то же время, они могут быть полезными инструментами для началь-

ного обучения и понимания основных концепций нейронных сетей. В целом, однослойные искусственные нейронные сети представляют собой важный элемент в области машинного обучения и искусственного интеллекта, но их применение ограничено и требует дальнейшего развития и улучшения.

Простейшая сеть состоит из группы нейронов, образующих слой, как показано в правой части рисунка 5.

Отметим, что вершины-круги слева служат лишь для распределения входных сигналов. Они не выполняют каких-либо вычислений и, поэтому не будут считаться слоем. Для большей наглядности обозначим их кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов X отдельным весом соединен с каждым искусственным нейроном. А каждый нейрон выдает взвешенную сумму входов в сеть.

В искусственных и биологических сетях многие соединения могут отсутствовать, но здесь они показаны все для демонстрации общей картины. Могут существовать также соединения между выходами и входами элементов в слое.

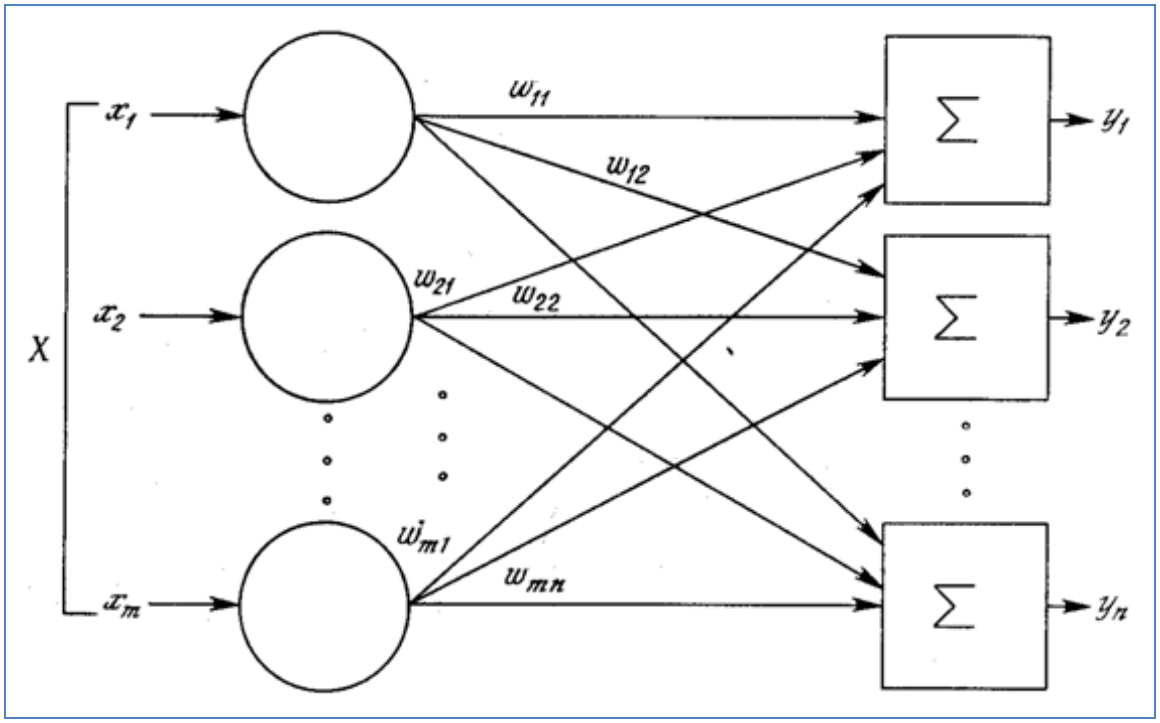

Рисунок 5 – Простейшая сеть из группы нейронов

Удобно считать веса элементами матрицы W. Матрица имеет m строк и n столбцов, где m – число входов, а n – число нейронов. Например, $w_{2,3}$ – это вес, связывающий второй вход с третьим нейроном. Таким образом, вычисление выходного вектора N, компонентами которого являются выходы OUT нейронов, сводится к матричному умножению $N = XW$, где N и X – векторы-строки.

Многослойные искусственные нейронные сети

Многослойные искусственные нейронные сети являются мощным инструментом в области машинного обучения. Они представляют собой сеть из нескольких слоев нейронов, которые взаимодействуют между собой, передавая и обрабатывая информацию. Каждый слой может содержать различное количество нейронов, и каждый нейрон имеет свои веса, которые определяют его важность при обработке входных данных.

Основное преимущество многослойных искусственных нейронных сетей заключается в их способности обучаться на основе предоставленных обучающих данных. Путем подачи входных данных на вход сети и последующего сравнения результата с ожидаемым, сеть корректирует веса нейронов и настраивает свою структуру для достижения более точных предсказаний. Это позволяет сети обучаться на основе опыта и улучшать свою производительность с течением времени.

Благодаря своей сложной структуре и возможности обучения, многослойные искусственные нейронные сети способны решать разнообразные задачи, включая распознавание образов, классификацию данных, анализ текстов и многое другое. Они нашли применение в таких областях, как медицина, финансы, робототехника и многие другие.

Однако, несмотря на свою мощность и гибкость, многослойные искусственные нейронные сети имеют и свои ограничения. Они требуют большого количества данных для обучения, и их обучение может быть длительным и ресурсоемким процессом. Кроме того, сети могут страдать от проблемы переобучения, когда они слишком точно запоминают обучающие данные и не способны обобщить свои знания на новые ситуации.

В целом, многослойные искусственные нейронные сети представляют собой важный инструмент в области машинного обучения и

имеют широкий спектр применения. Они продолжают развиваться и совершенствоваться, открывая новые возможности в решении сложных задач и улучшении производительности систем и приложений.

Подобная сеть показана на рисунке 6 и снова изображена со всеми соединениями.

Если между слоями многослойной сети используется линейная активационная функция, то она не приведет к повышению вычислительной мощности по сравнению с однослойной сетью. В таком случае, вычисление выходного значения слоя сводится к последовательному перемножению входного вектора на первую весовую матрицу, а затем умножению полученного результата на вторую весовую матрицу, при отсутствии нелинейной активационной функции.

$$OUT = (XW_1)W_2.$$

Так как умножение матриц ассоциативно, то

$$(XW_1)W_2 = X(W_1W_2).$$

Доказательство предполагает, что двухслойная линейная сеть эквивалентна одному слою, если весовая матрица равна двум весовым матрицам. Многослойные линейные сети очень устойчивы, так как они могут быть заменены одноуровневой сетью без каких-либо проблем. Одноуровневые сети очень сложные с вычислительной точки зрения, несмотря на их существование в однослойных сетях. Способность сетей выходить за пределы однослойной сети затрудняется функцией нелинейной активации, которая может быть использована для их расширения.

Несмотря на то, что эти сети были исследованы, в них отсутствовали определенные обратные, т.е. слоистые выходы, представляют собой входы от выходов предыдущих слоев или от выходов некоторого слоя к входам того же слоя или соседних слоев, что приводит к соединениям на выходах предыдущего слоя или от выходов определенного слоя. Сети без обратной связи или сети прямого распределения – оба типа сетей, которые очень интересны и обычно использу-

ются. Сети обратной связи представляют собой более широкие сети, которые имеют выходы и входы, связанные с ними, которые известны как сети обратной связи. Сети, в которых отсутствует обратная связь, не имеют памяти, в результате чего их выход полностью зависит от текущих входных данных и значений веса, включая продолжительность времени и пространства.

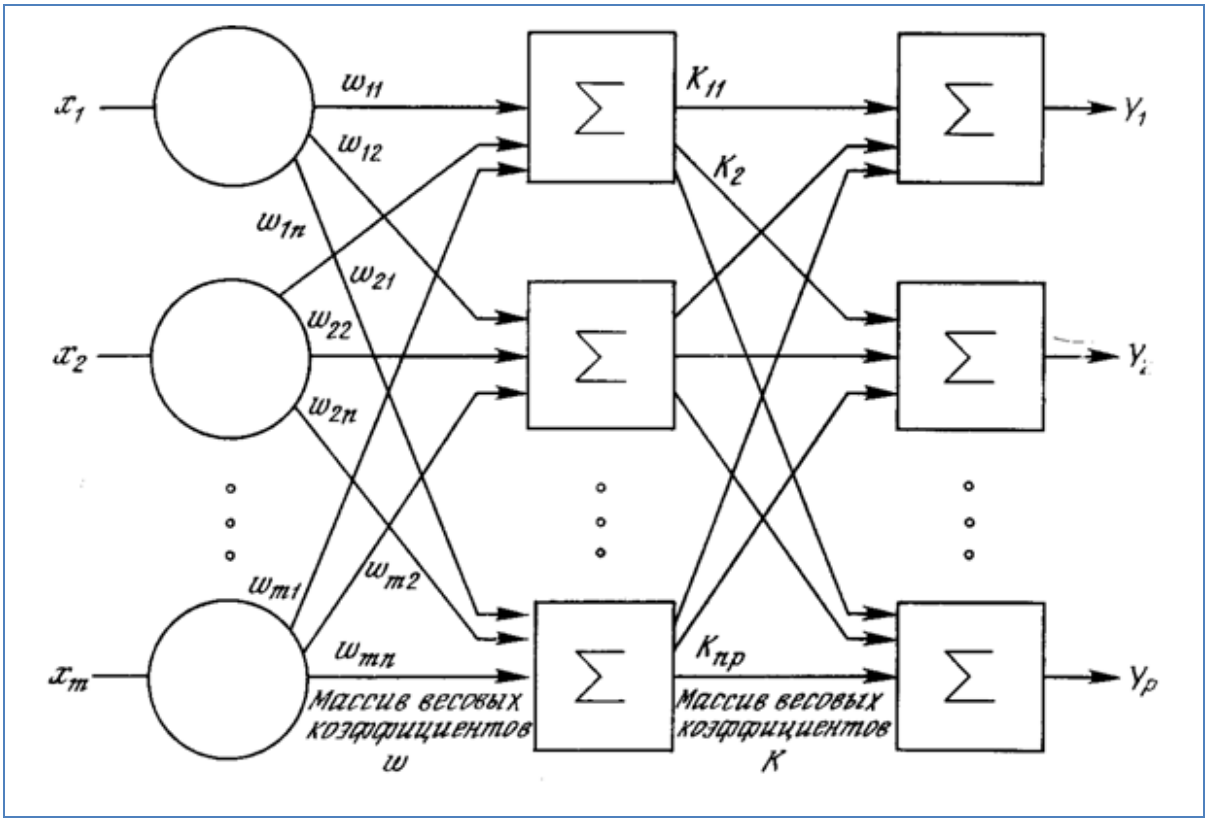

Рисунок 6 – Многослойная искусственная нейронная сеть

В некоторых сетевых конфигурациях с обратными линиями предыдущие выходные значения возвращаются на входы, в результате чего выход определяется как текущий вход, так и предыдущий выход. Краткосрочная человеческая память может быть сродни сетям обратных ссылок, где на сетевые выходы могут влиять предыдущие входы, что приводит к сетям на основе обратных линий.

К сожалению, нет общепринятого способа подсчета числа слоев в сети. Многослойная сеть состоит, как показано на рисунке 6, из чередующихся множеств нейронов и весов. Ранее, в связи с рисунком 5, уже говорилось, что входной слой не выполняет суммирования. Эти

нейроны служат лишь в качестве разветвлений для первого множества весов и не влияют на вычислительные возможности сети. По этой причине первый слой не принимается во внимание при подсчете слоев, и сеть, подобная изображенной на рисунке 6, считается двуслойной, так как только два слоя выполняют вычисления. Далее, веса слоя считаются связанными со следующими за ними нейронами. Следовательно, слой состоит из множества весов со следующими за ними нейронами, суммирующими взвешенные сигналы.

Персептроны и зарождение искусственных нейронных сетей

Первое систематическое изучение искусственных нейронных сетей было предпринято Маккалокком и Питтсом в 1943 г. Позднее они исследовали сетевые парадигмы для распознавания изображений, подвергаемых сдвигам и поворотам. Простая нейронная модель, показанная на рисунке 7, использовалась в большей части их работ. Элемент $\sum$ умножает каждый вход x на вес w и суммирует взвешенные входы. Если полученная сумма больше заданного порогового значения, выход равен единице, в противном случае – нулю. Эти системы (и множество им подобных) получили название *персептронов*.

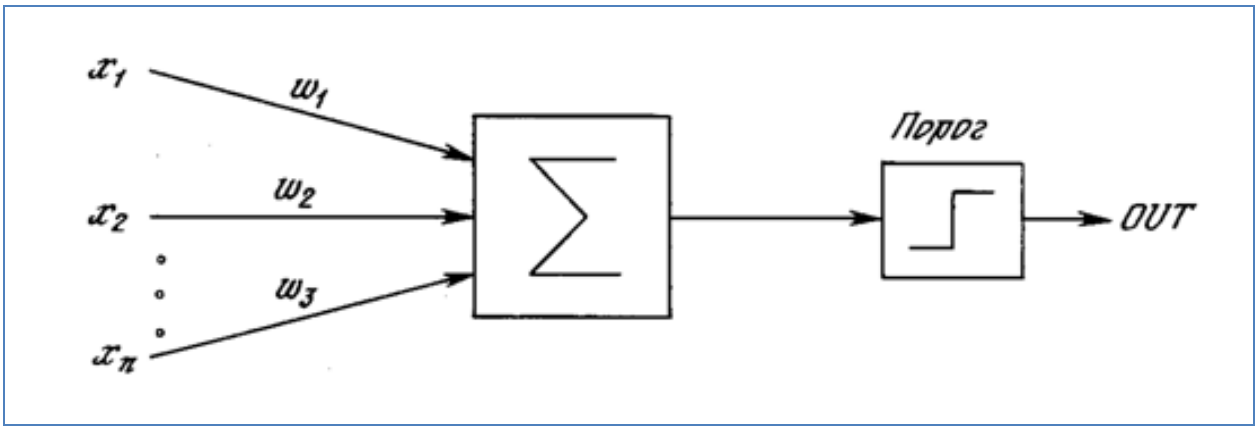

Рисунок 7 – Простейшая нейронная модель персептрона

Персептрон – это модель искусственного нейрона, который является основным строительным блоком искусственных нейронных сетей. Он основан на простой идеи биологического нейрона, который принимает входные сигналы, обрабатывает их и генерирует выходной сигнал.

Персептрон имеет несколько входов, каждый из которых имеет свой вес. Входной сигнал умножается на соответствующий вес, а затем суммируется. Затем, сумма проходит через функцию активации, которая определяет, должен ли нейрон выдать активацию или нет. Функция активации может быть разной – сигмоидальная, гиперболический тангенс, ReLU и другие.

Один персептрон может быть использован для решения простых задач классификации. Однако, чтобы решать более сложные задачи, необходимо объединить несколько персептронов в виде нейронной сети. Такие нейронные сети являются основой для глубокого обучения и могут успешно решать задачи распознавания образов, обработки естественного языка, компьютерного зрения и многих других.

Персептроны являются ключевым элементом в зарождении искусственных нейронных сетей. Они являются простыми моделями искусственных нейронов, которые могут быть объединены в более сложные нейронные сети для решения различных задач. Благодаря своей гибкости и мощности, искусственные нейронные сети находят широкое применение во многих областях, и будущее искусственного интеллекта связано с их развитием и усовершенствованием.

Они состоят из одного слоя искусственных нейронов, соединенных с помощью весовых коэффициентов с множеством входов (рис. 8), хотя, в принципе, описываются и более сложные системы.

В 60-е годы персептроны вызвали большой интерес и оптимизм. Одной из первых искусственных сетей, способных к перцепции (восприятию) и формированию реакции на воспринятый раздражитель, явился PERCEPTRON Розенблатта (F.Rosenblatt, 1957). Персептрон рассматривался его автором не как конкретное техническое (вычислительное) устройство, а как модель работы мозга. Розенблатт называл такую нейронную сеть трехслойной, однако, по современной терминологии, представленная сеть обычно называется *однослойной*, так как имеет только один слой нейропроцессорных элементов.

Ф. Розенблатт доказал замечательную теорему об обучении персептронов.

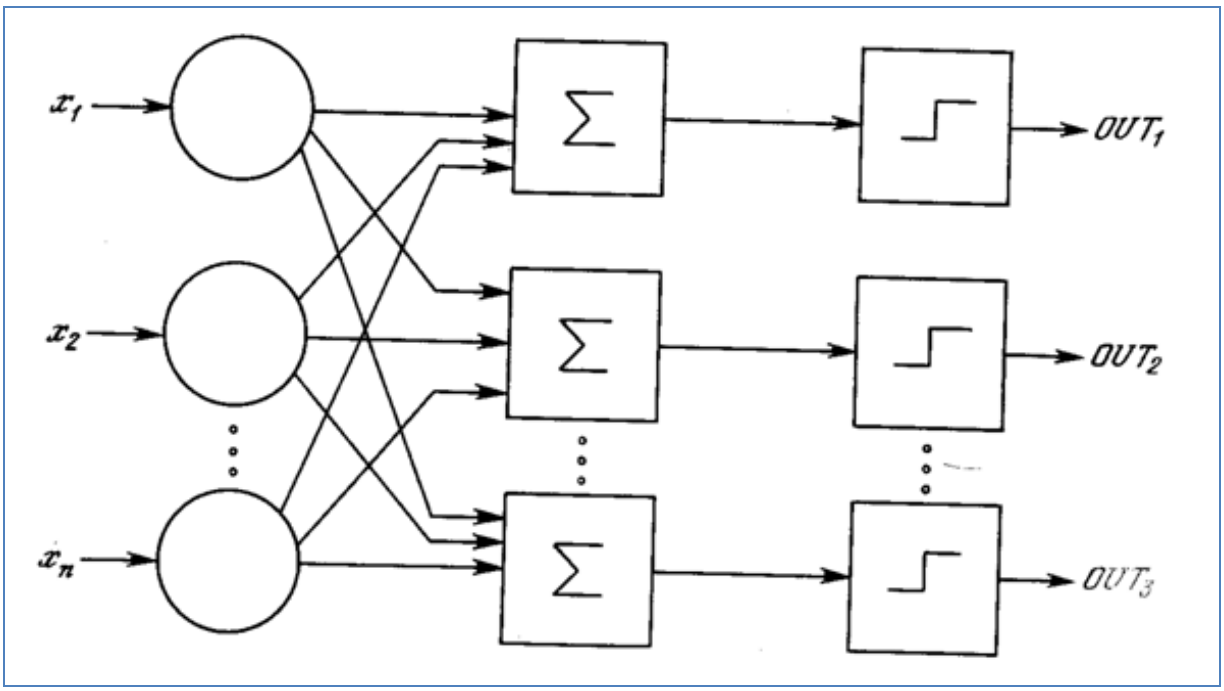

Рисунок 8 – Персептрон

Д. Уидроу дал ряд убедительных демонстраций систем персептронного типа, и исследователи во всем мире стремились изучить возможности этих систем. Первоначальная эйфория сменилась разочарованием, когда оказалось, что персептроны не способны обучаться решению ряда простых задач.

М.Л. Минский проанализировал эту проблему и показал, что имеются жесткие ограничения того, что могут выполнять однослойные персептроны, и, следовательно, того, чему они могут обучаться. Так как в то время методы обучения многослойных сетей не были известны, исследователи занялись более многообещающими проектами, и исследования в области нейронных сетей пришли в упадок. Недавнее открытие методов обучения многослойных сетей привело к возрождению интереса и возобновлению исследований.

Доказательство теоремы обучения персептрона показало, что персептрон способен научиться всему, что он способен представлять. Важно при этом уметь различать представляемость и обучаемость. Понятие представляемости относится к способности персептрона (или другой сети) моделировать определенную функцию. Обучаемость же требует наличия систематической процедуры настройки весов сети для реализации этой функции.

2. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

Искусственные нейронные сети (ИНС) обладают замечательной способностью к обучению, что является одним из наиболее значимых аспектов их существования. ИНС обучаются с использованием различных методов, но большинство из них основаны на общих принципах и имеют схожие характеристики. Данная работа рассматривает некоторые фундаментальные алгоритмы обучения ИНС, включая их современную применимость и историческую значимость. Понимание этих основных алгоритмов позволяет легче постичь другие алгоритмы, а также лучше понять и развивать новые разработки в этой области.

Обучающие алгоритмы ИНС классифицируются на два типа: обучение с учителем и обучение без учителя.

Обучение с учителем

При обучении с учителем (также известном как контролируемое обучение) ИНС получает входные данные и желаемые выходные данные от преподавателя или внешнего источника. Затем сеть настраивает свои веса, чтобы минимизировать разницу между фактическими и желаемыми выходными данными. Этот процесс продолжается, пока сеть не достигнет желаемой производительности. Методы обучения с учителем включают:

– Обратное распространение ошибки (Backpropagation): Является одним из наиболее распространённых и универсальных алгоритмов обучения ИНС. В процессе обучения выход сети сравнивается с желаемыми выходными данными, и ошибка вычисляется как разница между ними. Затем эта ошибка распространяется обратно через сеть, и веса нейронов настраиваются таким образом, чтобы уменьшить ошибку.

– Алгоритм Левенберга-Марквардта (Levenberg-Marquardt): Адаптивный алгоритм градиентного типа, который использует информацию о кривизне целевой функции для более быстрого схождения, чем обратное распространение ошибки.

– Метод Ньютона (Newton's Method): Использует вторую производную целевой функции для ускорения схождения алгоритма обучения.

Обучение без учителя

При обучении без учителя (называемом также неконтролируемым обучением) ИНС получает только входные данные и должна самостоятельно находить структуру или взаимосвязи в этих данных. Этот тип обучения не предполагает наличия желаемых выходных данных, поэтому сеть должна самостоятельно определять, что является важным в данных и как их организовать. Методы обучения без учителя включают:

– Самоорганизующиеся карты (Self-Organizing Maps, SOM): Метод обучения, который проецирует высокоразмерные данные в низкоразмерное пространство, сохраняя топологические отношения между данными.

– Алгоритм k-средних (k-Means Clustering): Один из наиболее популярных алгоритмов кластеризации, который разделяет данные на заданное количество кластеров на основе расстояния между точками данных.

– Алгоритм декомпозиции (Principal Component Analysis, PCA): Линейный преобразователь, который преобразует коррелированные данные в набор ортогональных переменных, называемых главными компонентами.

Современные ИНС используются во многих областях, включая распознавание образов, обработку естественного языка, машинное обучение и другие. Прогресс в области обучения ИНС позволил создавать нейронные сети, которые достигают впечатляющих результатов в различных задачах, таких как классификация изображений, перевод языков и голосовое управление. Однако, несмотря на значительные достижения, обучение ИНС остаётся сложной задачей, и исследователи продолжают работать над разработкой новых и более эффективных алгоритмов обучения.

Метод дифференциального обучения Хэбба

Метод сигнального обучения Хэбба предполагает вычисление свертки предыдущих изменений выходов для определения изменения весов. Данный же метод, называемый методом дифференциального обучения Хэбба, использует следующее равенство:

$$w_{ij}(t+1) = w_{ij}(t) + [OUT_i(t) - OUT_i(t-1)][OUT_j(t) - OUT_j(t-1)],$$

где $w_{ij}(t)$ – сила синапса от нейрона i к нейрону j в момент времени t, $OUT_i(t)$ – выходной уровень пресинаптического нейрона в момент времени t, $OUT_j(t)$ – выходной уровень постсинаптического нейрона в момент времени t.

Входные и выходные звезды

В области искусственных нейронных сетей работы С. Гроссберга заложили основу для многих ключевых идей и архитектур. Среди них особое место занимают концепции входных и выходных звезд, являющихся фундаментальными компонентами многих сетевых парадигм.

Входная звезда (рис. 9) состоит из единичного нейрона, принимающего массив входов через синаптические веса. Основная задача входной звезды заключается в выполнении распознавания образов, то есть она обучается реагировать на определенный входной вектор X и игнорировать все остальные. Этот процесс обучения осуществляется путем тонкой настройки весов в соответствии с характеристиками входного вектора.

Выход входной звезды определяется взвешенной суммой ее входов, что можно представить как свертку входного вектора с весовым вектором. Такой подход позволяет нейрону выявлять сходство между входным образом и эталонным образцом, в соответствии с которым он был обучен. Таким образом, нейрон демонстрирует максимальную реакцию на те входные данные, которые наиболее близки к изученному эталонному образцу.

В отличие от входных звезд, выходные звезды (рис. 10) обладают дополнительной функцией: они генерируют требуемые возбуждающие сигналы для других нейронов при собственной активации. Обучение выходной звезды также осуществляется путем настройки весов в соответствии с целевым вектором. Подобно входным звездам, веса выходной звезды постепенно корректируются путем обработки набора векторов, представляющих типичные вариации идеального вектора. В результате выходной сигнал нейрона становится статистической характеристикой обучающего набора и имеет тенденцию приближаться к идеальному вектору при обработке искаженных версий исходных данных в процессе обучения.

Входные и выходные звезды могут быть взаимно соединены в сети любой сложности; Гроссберг рассматривает их как модель

определенных биологических функций. Вид звезды определяет ее название, однако, звезды обычно изображаются в сети несколько иначе.

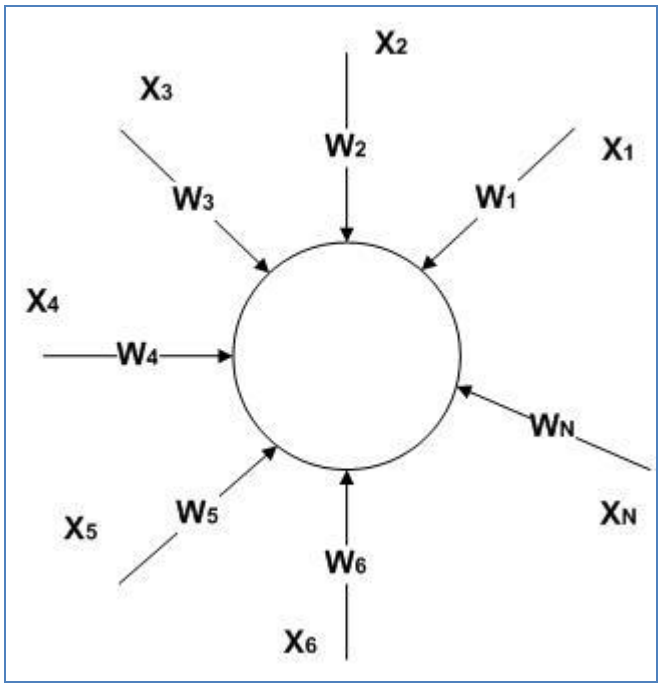

Рисунок 9 – Модель входной звезды

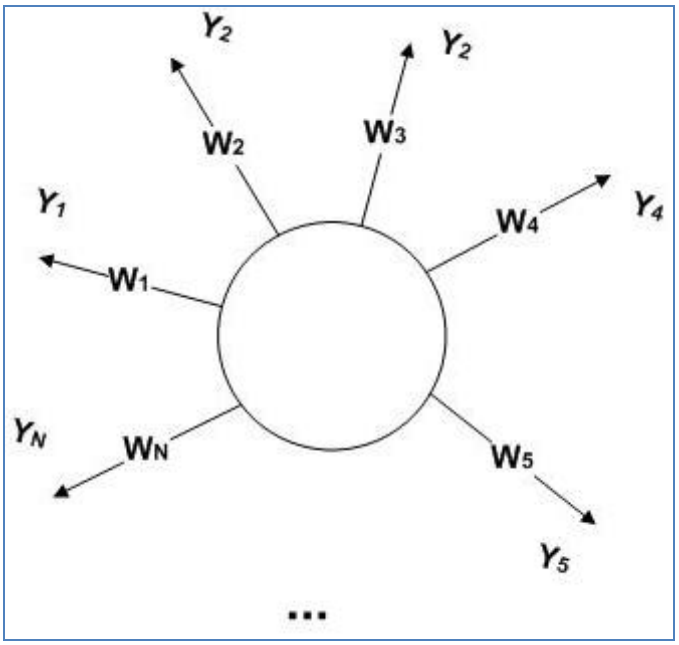

Рисунок 10 – Модель выходной звезды

Входные и выходные звезды могут быть соединены различными способами для создания сетей разной сложности и функциональности. Часто используемые конфигурации включают:

1) Сетевую архитектуру с прямой связью: простая последовательность входных и выходных звезд, где выход каждой входной звезды является входом для соответствующей выходной звезды. Эта базовая архитектура используется для задач классификации и прогнозирования.

2) Сетевую архитектуру с обратной связью: в сетях с обратной связью выходные звезды влияют на входные звезды через дополнительные соединения, образуя замкнутый цикл обработки информации. Эта архитектура применяется в динамических системах и задачах обучения с учителем.

3) Многослойные сети: в многослойных сетях входные и выходные звезды организованы в несколько слоев, позволяя моделировать более сложные функции и отношения между данными. Такие сети обычно используются в глубоком обучении и различных приложениях искусственного интеллекта.

Входные и выходные звезды играют решающую роль в функционировании нейронных сетей. Их способность распознавать входные данные, генерировать выходные сигналы и объединяться в различные сетевые архитектуры делает их важными строительными блоками для широкого спектра приложений, включая обработку изображений, распознавание речи, машинное обучение и автономное управление. Благодаря своей гибкости и широким возможностям звездные конфигурации остаются основой многих современных нейронных сетей, обеспечивая их эффективность и универсальность.

Обучение входной звезды

Входная звезда выполняет распознавание образов, т.е. она обучается реагировать на определенный входной вектор X и ни на какой другой. Это обучение реализуется, настраивая веса таким образом, чтобы они соответствовали входному вектору. Выход входной звезды определяется как взвешенная сумма ее входов, это описано в предыдущих разделах. С другой точки зрения, выход можно рассматривать как свертку входного вектора с весовым вектором или меру сходства нормализованных векторов.

Следовательно, нейрон должен реагировать наиболее сильно на входной образ, которому был обучен.

Процесс обучения выражается следующим образом:

$$w_i(t+1) = w_i(t) + \alpha[x_i - w_i(t)],$$

где w_i – вес входа x_i, x_i – i-й вход, α – нормирующий коэффициент обучения, который имеет начальное значение 0,1 и постепенно уменьшается в процессе обучения.

После завершения обучения предъявление входного вектора X будет активизировать обученный входной нейрон. Это можно рассматривать как единый обучающий цикл, если α установлен в 1, однако в этом случае исключается способность входной звезды к обобщению. Хорошо обученная входная звезда будет реагировать не только на определенный единичный вектор, но также и на незначительные изменения этого вектора. Это достигается постепенной настройкой нейронных весов при предъявлении в процессе обучения векторов, представляющих нормальные вариации входного вектора. Веса настраиваются таким образом, чтобы усреднить величины обучающих векторов, и нейроны получают способность реагировать на любой вектор этого класса.

Обучение выходной звезды

В то время как входная звезда возбуждается всякий раз при появлении определенного входного вектора, выходная звезда имеет дополнительную функцию: она вырабатывает требуемый возбуждающий сигнал для других нейронов всякий раз, когда возбуждается.

Для того чтобы обучить нейрон выходной звезды, его веса настраиваются в соответствии с требуемым целевым вектором. Алгоритм обучения может быть представлен символически следующим образом:

$$w_i(t+1) = w_i(t) + \beta[y_i - w_i(t)],$$

где β представляет собой нормирующий коэффициент обучения, который вначале приблизительно равен единице и постепенно уменьшается до нуля в процессе обучения.

Как и для входной звезды, веса выходной звезды постепенно настраиваются над множеством векторов, представляющих собой обычные вариации идеального вектора. В этом случае выходной сигнал нейронов является статистической характеристикой обучающего набора и может в действительности сходиться в процессе обучения к идеальному вектору при предъявлении только искаженных версий вектора.

Обучение персептрона

«Персептрон должен решать задачу классификации по бинарным входным сигналам. Набор входных сигналов будем обозначать m-мерным вектором x. Все элементы вектора являются булевыми переменными (переменными, принимающими значения Истина или Ложь). Однако иногда полезно оперировать числовыми значениями. Будем считать, что значению «ложь» соответствует числовое значение 0, а значению «истина» соответствует 1.

Напомним, что персептроном будем называть устройство, вычисляющее следующую систему функций (3) » [2]:

$$\psi = \left[\sum_{i=1}^{m} w_i x_i > \theta\right],$$

Где w_i – веса персептрона, θ – порог, x_i – значения входных сигналов, скобки [] означают переход от булевых (логических) значений к числовым значениям по правилам, изложенным выше.

«Обучение персептрона состоит в подстройке весовых коэффициентов. Пусть имеется набор пар векторов (x^α, y^α), $\alpha = 1, \ldots, p$, называемый обучающей выборкой. Будем называть нейронную сеть обученной на данной обучающей выборке, если при подаче на входы сети каждого вектора x^α на выходах всякий раз получается соответствующий вектор y^α» [2].

«Предложенный Ф.Розенблаттом метод обучения состоит в итерационной подстройке матрицы весов, последовательно уменьшающей ошибку в выходных векторах. Алгоритм включает несколько шагов» [2]:

Шаг 0	Начальные значения весов всех нейронов $W(t=0)$ полагаются случайными
Шаг 1	Сети предъявляется входной образ x^{α}, в результате формируется выходной образ $\tilde{y}^{\alpha} \neq y^{\alpha}$.
Шаг 2	Вычисляется вектор ошибки $\delta^{\alpha} = (y^{\alpha} - \tilde{y}^{\alpha})$, делаемой сетью на выходе. Дальнейшая идея состоит в том, что изменение вектора весовых коэффициентов в области малых ошибок должно быть пропорционально ошибке на выходе и равно нулю, если ошибка равна нулю.
Шаг 3	Вектор весов модифицируется по следующей формуле: $W(t + \Delta T = W(t) + \eta x^{\alpha} \cdot (\delta^{\alpha})^{T}$. Здесь $0 < \eta < 1$ – темп обучения.
Шаг 4	Шаги 1–3 повторяются для всех обучающих векторов. Один цикл последовательного предъявления всей выборки называется эпохой. Обучение завершается по истечении нескольких эпох: а) когда итерации сойдутся, т.е. вектор весов перестает изменяться, или б) когда полная, просуммированная по всем векторам абсолютная ошибка станет меньше некоторого малого значения.

«Объясним данный алгоритм более подробно. Подаем на вход персептрона такой вектор x, для которого уже известен правильный ответ. Если выходной сигнал персептрона совпадает с правильным ответом, то никаких действий предпринимать не надо. В случае ошибки, необходимо обучить персептрон правильно решать данный пример. Ошибки могут быть двух типов. Рассмотрим каждый из них» [2].

Если на выходе персептрона получен ноль, а правильный ответ равен одному, это означает, что произошла ошибка. Чтобы исправить эту ошибку, необходимо увеличить сумму в правой части уравнения. Для этого можно увеличить веса связей между нейронами, которые активны одновременно. Важно отметить, что нет смысла увеличивать веса при переменных, которые равны нулю, поэтому следует концентрироваться на увеличении весов при переменных, которые равны единице. Это первое правило, которое поможет улучшить работу персептрона.

Выходной персептрон считается активным в данной ситуации. Ошибка второго типа возникает, когда выход персептрона равен 1, а правильный ответ равен нулю. Чтобы исправить эту ошибку, необходимо уменьшить сумму в правой части уравнения. Для этого следует уменьшить веса связей переменных, которые равны 1, так как нет смысла уменьшать веса связей при переменных, равных нулю (x_i). Также необходимо применить эту процедуру ко всем активным нейронам предыдущих слоев. В итоге получаем второе правило.

В процессе обучения мы последовательно перебираем все примеры из обучающего множества и применяем правила обучения для исправления ошибок. Если после каждого цикла предъявления примеров мы обнаруживаем, что все они решены правильно, то процедура обучения завершается.

«Нерассмотренными остались два вопроса. Первый – о сходимости процедуры обучения. Второй – на сколько нужно увеличивать (уменьшать) веса связей при применении правил обучения.

Ответ на первый вопрос дают следующие теоремы.

Теорема о сходимости персептрона. Если существует вектор параметров w, при котором персептрон правильно решает все примеры обучающей выборки, то при обучении персептрона по вышеописанному алгоритму решение будет найдено за конечное число шагов.

Теорема о «зацикливании» персептрона. Если не существует вектора параметров w, при котором персептрон правильно решает все примеры обучающей выборки, то при обучении персептрона по данному правилу через конечное число шагов вектор весов начнет повторяться.

Таким образом, данные теоремы утверждают, что, запустив процедуру обучения персептрона, через конечное время мы либо получим обучившийся персептрон, либо ответ, что данный персептрон поставленной задаче обучиться не может» [2].

Доказательства этих теорем в данное учебное пособие не включены.

«Целочисленность весов персептронов

Для ответа на вопрос о количественных характеристиках вектора *w* рассмотрим следующую теорему.

Теорема. Любой персептрон можно заменить другим персептроном того же вида с целыми весами связей.

Доказательство. Обозначим множество примеров одного класса (правильный ответ равен 0) через X_0, а другого (правильный ответ равен 1) – через X_1. Вычислим максимальное и минимальное значения суммы в правой части (3):

$$s_0 = \max_{x \in X_0} \sum_i w_i x_i,$$

$$s_1 = \min_{x \in X_1} \sum_i w_i x_i.$$

Определим допуск s как минимум из s_0 и s_1. Положим $\delta = s/(m+1)$, где m – число слагаемых в (3). Поскольку персептрон (3) решает поставленную задачу классификации и множество примеров в обучающей выборке конечно, то $\delta > 0$. Из теории чисел известна теорема о том, что любое действительное число можно сколь угодно точно приблизить рациональными числами. Заменим веса w_i на рациональные числа так, чтобы выполнялись следующие неравенства: $|w_i - w_i'| < \delta$.» [2]

«Из этих неравенств следует, что при использовании весов w_i' персептрон будет работать с теми же результатами, что и первоначальный персептрон. Действительно, если правильным ответом примера является 0, имеем $\sum_i w_i x_i \le -s$.

Подставив новые веса, получим:

$$\begin{aligned} \sum_i w_i' x_i &= \sum_i (w_i' - w_i) x_i + \sum_i w_i x_i \le \\ &\le \sum_i |w_i' - w_i| x_i - s \le \\ &\le \sum_i |w_i' - w_i| - s < (m+1)\delta - s = 0. \end{aligned}$$

Откуда следует необходимое неравенство (4)

$$\sum_i w_i' x_i < 0.$$

Аналогично, в случае правильного ответа равного 1, имеем $\sum_i w_i x_i < s$, откуда, подставив новые веса и порог, получим:

$$\sum_i w_i' x_i = \sum_i (w_i' - w_i) x_i + \sum_i w_i x_i \geq s - \sum_i |w_i' - w_i| x_i \geq \geq s - \sum_i |w_i' - w_i| > s - (m+1)\delta = 0.$$

Отсюда следует выполнение неравенства (5)

$$\sum_i w_i' x_i > 0.$$

Неравенства (4) и (5) доказывают возможность замены всех весов и порога любого персептрона рациональными числами. Очевидно также, что при умножении всех весов и порога на одно и то же ненулевое число персептрон не изменится. Поскольку любое рациональное число можно представить в виде отношения целого числа к натуральному числу, получим (6)

$$\psi = \left[\sum_{i=1}^{m} w_i x_i > 0\right] = \left[\sum_{i=1}^{m} w_i' x_i > 0\right] = \left[\sum_{i=1}^{m} \frac{w_i''}{r_i} x_i > 0\right]$$

где w_i' – целые числа. Обозначим через r произведение всех знаменателей:

$$r = \prod_{i=0}^{m} r_i$$

Умножим все веса и порог на r Получим веса целочисленные $w''' = rw'''$.

Из (4), (5) и (6) получаем

$$\psi = \left[\sum_{i=1}^{m} w_i x_i > 0\right] = \left[\sum_{i=1}^{m} w_i' x_i > 0\right] = \left[\sum_{i=1}^{m} \frac{w_i''}{r_i} x_i > 0\right] =$$
$$= \left[\sum_{i=1}^{m} w''' x_i > 0\right],$$

что и завершает доказательство теоремы» [2].

«Поскольку из доказанной теоремы следует, что веса персептрона являются целыми числами, то вопрос о выборе шага при применении правил обучения решается просто: веса и порог следует увеличивать (уменьшать) на единицу.

Как уже упоминалось, алгоритм обучения персептрона возможно использовать и для многослойных персептронов. Однако теоремы о сходимости и зацикливании персептрона, приведенные выше, верны только при обучении однослойного персептрона – или многослойного персептрона при условии, что обучаются только веса персептрона, стоящего в последнем слое сети. В случае произвольного многослойного персептрона они не работают. Следующий пример демонстрирует основную проблему, возникающую при обучении многослойных персептронов.

Пусть веса всех слоев персептрона в ходе обучения сформировались так, что все примеры обучающего множества, кроме первого, решаются правильно. При этом правильным ответом первого примера является 1. Все входные сигналы персептрона последнего слоя равны нулю. В этом случае первое правило не дает результата, поскольку все нейроны предпоследнего слоя не активны. Существует множество способов решать эту проблему. Однако все эти методы не являются регулярными и не гарантируют сходимость многослойного персептрона к решению, даже при условии, что такое решение существует» [2].

В действительности, проблема настройки (обучения) многослойного персептрона решается следующей теоремой.

«Теорема о двуслойности персептрона. Любой многослойный персептрон может быть представлен в виде двуслойного персептрона с необучаемыми весами первого слоя.

Для доказательства этой теоремы потребуется одна теорема из математической логики.

Теорема о дизъюнктивной нормальной форме. Любая булева функция булевых аргументов может быть представлена в виде дизъюнкции конъюнкций элементарных высказываний и отрицаний элементарных высказываний:

$$f = \vee \left(\&x_i \& \neg x_j\right).$$

Напомним некоторые свойства дизъюнктивной нормальной формы.

Свойство 1. В каждый конъюнктивный член (слагаемое) входят все элементарные высказывания либо в виде самого высказывания, либо в виде его отрицания.

Свойство 2. При любых значениях элементарных высказываний в дизъюнктивной нормальной форме может быть истинным не более одного конъюнктивного члена (слагаемого)» [2].

«Доказательство теоремы о двуслойности персептрона. Из теоремы о дизъюнктивной нормальной форме следует, что любой многослойный персептрон может быть представлен в следующем виде (7):

$$\psi = [\vee\left(\&x_i \& \neg x_j\right)]$$

В силу второго свойства дизъюнктивной нормальной формы, равенство (7) можно переписать в виде (8)

$$\psi = \left[\vee\left(\&x_i \& \neg x_j\right)\right] = \left[\sum\left[\left(\&x_i \& \neg x_j\right)\right] > 0\right]$$

Переведем в арифметическую форму все слагаемые в выражении (8). Конъюнкцию заменяем на умножение, а отрицание на разность: $\neg x_j = 1 - x_j$. Произведя эту замену и приведя подобные члены, получим (9):

$$\psi = \left[\sum_{l} \alpha_l \prod_{i \in I_l} x_i > 0 \right],$$

где I_l – множество индексов сомножителей в m- слагаемом, α_l – число, указывающее, сколько раз такое слагаемое встретилось в выражении (8) после замены и раскрытия скобок (число подобных слагаемых).

Заменим -е слагаемое в формуле (9) персептроном следующего вида (10):

$$\varphi_i = \prod_{l \in I_l} x_l = \left[\sum_{l \in I_l} x_l > |I_l| - 1 \right].$$

Подставив выражение (10) в формулу (9), получим равенство (3), то есть произвольный многослойный персептрон представлен в виде (3) с целочисленными коэффициентами. В качестве персептронов первого слоя используются персептроны вида (10) с необучаемыми весами. Теорема доказана» [2].

«Подводя итоги, следует отметить следующие основные свойства персептронов:

1. Любой персептрон может содержать один или два слоя. В случае двухслойного персептрона веса первого слоя не обучаются.
2. Веса любого персептрона можно заменить на целочисленные.
3. При обучении после конечного числа итераций возможны два исхода: персептрон обучится или вектор весов персептрона будет повторяться (персептрон зациклится).

Знание этих свойств позволяет избежать «усовершенствований» типа модификации скорости обучения и других, столь же «эффективных» модернизаций» [2].

«Трудности с алгоритмом обучения персептрона

Иногда бывает сложно определить, выполнено ли условие разделимости для конкретного обучающего множества. Кроме того, во многих встречающихся на практике ситуациях входы часто меняются во времени и могут быть разделимы в один момент времени и неразделимы – в другой. В доказательстве алгоритма обучения персептро-

на ничего не говорится также о том, сколько шагов требуется для обучения сети. Мало утешительного знать, что обучение закончится за конечное число шагов, если необходимое для этого время сравнимо с геологической эпохой. Кроме того, не доказано, что персептронный алгоритм обучения более быстр по сравнению с простым перебором всех возможных значений весов, и в некоторых случаях этот примитивный подход может оказаться лучше. Эти проблемы являются важной областью современных исследований» [2].

Метод обучения Уидроу-Хоффа

Метод обучения Уидроу-Хоффа, также известный как метод нейронных сетей с обратным распространением ошибки, является одним из основных методов машинного обучения. Он представляет собой алгоритм, который позволяет нейронной сети обучаться на основе входных данных и соответствующих им выходных значений.

Основная идея метода Уидроу-Хоффа заключается в том, чтобы определить веса связей между нейронами таким образом, чтобы минимизировать ошибку между выходом нейронной сети и ожидаемым выходом. Для этого используется градиентный спуск – алгоритм оптимизации функции, который позволяет находить минимум функции путем изменения ее параметров в направлении наискорейшего убывания.

Процесс обучения начинается с инициализации случайных значений весов связей между нейронами. Затем для каждого примера из тренировочного набора данных происходит прямое распространение через сеть: значения весов связей перемножаются на значения активаций предыдущего слоя и суммируются, после чего применяется функция активации нейрона. Это позволяет получить значения выходов нейронной сети.

Далее происходит расчет ошибки – разницы между ожидаемыми выходами и полученными значениями. Ошибка используется для нахождения градиента функции ошибки по весам связей нейронной сети. Затем веса обновляются в направлении антиградиента, что позволяет уменьшить ошибку.

Процесс обучения продолжается до тех пор, пока значение ошибки не станет достаточно малым или пока не будет достигнуто заданное количество эпох обучения. В результате обучения нейронная сеть настраивается на определенные закономерности в данных и

может использоваться для предсказания выходных значений для новых, ранее неизвестных примеров.

Однако метод Уидроу-Хоффа имеет свои ограничения и проблемы. Во-первых, он требует большого объема тренировочных данных для достижения хороших результатов. Недостаточный объем данных может привести к переобучению или недообучению модели.

Кроме того, метод Уидроу-Хоффа может страдать от проблемы исчезающего градиента. Это означает, что при обратном распространении ошибки градиент сети может сильно уменьшаться по мере прохождения через слои, что затрудняет обновление весов связей на начальных слоях.

Для решения этих проблем были предложены различные модификации метода Уидроу-Хоффа, такие как алгоритмы оптимизации (например, стохастический градиентный спуск), использование более сложных архитектур нейронной сети (например, рекуррентные или сверточные нейронные сети) и применение регуляризации для контроля переобучения.

Метод Уидроу-Хоффа является одним из фундаментальных методов машинного обучения и широко используется в различных областях, таких как компьютерное зрение, естественный языкообразование и распознавание речи. Понимание его работы и ограничений позволяет эффективно применять этот метод для решения практических задач.

Методы статистического обучения

Методы статистического обучения являются одними из основных методов машинного обучения. Они основаны на использовании статистических моделей и вероятностных методов для анализа данных и построения прогностических моделей.

Одним из ключевых понятий в статистическом обучении является понятие выборки. Выборка представляет собой набор данных, на основе которого будет строиться модель. Важно иметь достаточное количество данных для обучения модели, чтобы она была способна выявить закономерности и делать точные прогнозы.

Существует несколько различных методов статистического обучения, каждый из которых имеет свои особенности и применяется в различных ситуациях.

1. Линейная регрессия – это один из самых простых и широко используемых методов статистического обучения. Он используется

для анализа зависимости между переменными, где одна переменная (зависимая) зависит линейно от других переменных (независимых). Линейная регрессия позволяет предсказывать значения зависимой переменной на основе значений независимых переменных.

2. Логистическая регрессия – это метод статистического обучения, который используется для моделирования вероятности бинарного или категориального исхода. Он применяется в задачах классификации, где необходимо отнести объекты к определенным классам на основе значений независимых переменных.

3. Деревья решений – это метод статистического обучения, который представляет собой иерархическую структуру, состоящую из узлов и листьев. Каждый узел представляет собой тест на одну из переменных, а каждый лист представляет собой конечный результат или прогноз. Деревья решений могут использоваться как для задач классификации, так и для задач регрессии.

4. Метод k-ближайших соседей – это метод статистического обучения, который использует информацию о k ближайших точках в пространстве признаков для классификации или регрессии нового объекта. Он относит новый объект к тому классу или вычисляет его значение на основе значений ближайших соседей.

5. Наивный Байесовский классификатор – это метод статистического обучения, основанный на применении теоремы Байеса. Он предполагает независимость признаков и используется для задач классификации. Наивный Байесовский классификатор представляет собой вероятностную модель, которая вычисляет вероятности принадлежности объектов к различным классам.

Это лишь некоторые из методов статистического обучения, которые широко применяются в машинном обучении. Каждый метод имеет свои особенности и может быть эффективным в определенных ситуациях. Однако выбор метода зависит от конкретной задачи и доступных данных. Важно учитывать все факторы и выбрать наиболее подходящий метод для решения поставленной задачи.

Самоорганизация

Самоорганизация является одним из важных аспектов методов машинного обучения. Она позволяет системе самостоятельно организовываться и адаптироваться к изменяющейся среде, не требуя постоянного вмешательства со стороны человека.

В контексте методов машинного обучения, самоорганизация может быть реализована различными способами. Один из таких способов – автономное обучение. В этом случае, система самостоятельно определяет наиболее важные признаки и свойства данных, без участия человека. Это позволяет системе быстрее и эффективнее выполнять задачи классификации или прогнозирования.

Другой подход к самоорганизации – использование генетических алгоритмов. Генетический алгоритм основан на принципах естественного отбора и эволюции. Система создает набор случайных решений (которые представлены набором параметров), затем оценивает их по заданным критериям и сохраняет лучшие из них. Затем происходит «скрещивание» лучших решений, что позволяет получить новые комбинации параметров. Такой подход позволяет системе самостоятельно находить оптимальные решения без участия человека.

Кроме того, самоорганизация может быть достигнута с помощью нейронных сетей. Нейронная сеть – это математическая модель, имитирующая работу нервной системы человека. Она состоит из множества взаимосвязанных искусственных нейронов, которые передают и обрабатывают информацию. Нейронные сети способны самостоятельно «обучаться» на основе имеющихся данных и автоматически настраивать свои параметры для достижения оптимальной производительности.

Одним из примеров использования самоорганизации в методах машинного обучения является кластерный анализ. Кластерный анализ позволяет разбить данные на группы (кластеры) таким образом, чтобы объекты внутри одного кластера были более похожи друг на друга, чем на объекты из других кластеров. Алгоритм кластерного анализа может быть реализован с использованием различных методов самоорганизации, таких как иерархическая кластеризация или метод k-средних.

Однако, несмотря на все преимущества самоорганизации в методах машинного обучения, она также имеет свои ограничения. Например, при использовании автономного обучения, система может неправильно определить важность некоторых признаков данных или упустить значимую информацию. Поэтому важно проводить дополнительный анализ и проверять результаты работы системы.

Таким образом, самоорганизация является одним из ключевых аспектов методов машинного обучения. Она позволяет системе само-

стоятельно организовываться и адаптироваться к изменяющейся среде без участия человека. Различные подходы к самоорганизации, такие как автономное обучение, генетические алгоритмы или использование нейронных сетей, позволяют достичь оптимальных результатов в задачах классификации и прогнозирования. Однако необходимость дополнительной проверки результатов работы системы необходима для обеспечения точности и надежности.

Процедура обратного распространения

Процедура обратного распространения (backpropagation) является одним из основных алгоритмов обучения нейронных сетей. Она позволяет оптимизировать веса связей между нейронами, чтобы минимизировать ошибку предсказания модели.

Суть процедуры обратного распространения заключается в том, что ошибка на выходе сети распространяется обратно к входным слоям, корректируя значения весов связей между нейронами. Это позволяет моделировать сложные функции и учиться на основе примеров.

Процесс обратного распространения можно разделить на несколько этапов:

1. Прямое распространение: На этом этапе данные подаются на входной слой сети и передаются через все скрытые слои до выходного слоя. В каждом нейроне выполняется операция активации, которая преобразует суммированный входной сигнал нейрона в его выходное значение.

2. Вычисление ошибки: После получения предсказанных значений на выходном слое, происходит сравнение этих значений с желаемыми ответами из тренировочного набора данных. Разница между предсказанными и желаемыми значениями определяет ошибку модели.

3. Обратное распространение ошибки: На этом этапе происходит обратное распространение ошибки от выходного слоя к входному. Ошибка каждого нейрона вычисляется на основе его вклада в общую ошибку сети. Эта ошибка затем передается обратно через скрытые слои, учитывая веса связей между нейронами.

4. Корректировка весов: После вычисления ошибок для всех нейронов, происходит корректировка значений весов связей. Это делается путем изменения значений весов таким образом, чтобы минимизировать ошибку предсказания модели.

Важно отметить, что процедура обратного распространения требует градиентного спуска – метода оптимизации, который позволяет находить минимум функции ошибки путем последовательного изменения значений параметров модели.

Процедура обратного распространения является ключевым компонентом многих алгоритмов глубокого обучения, таких как многослойные перцептроны и сверточные нейронные сети. Она позволяет нейронным сетям обучаться на больших объемах данных и достигать высокой точности в предсказаниях.

Процедура обратного распространения является мощным инструментом для обучения нейронных сетей. Она позволяет моделировать сложные функции и учиться на основе примеров, корректируя значения весов связей между нейронами. Этот алгоритм играет ключевую роль в развитии и применении глубокого обучения для решения различных задач.

Обучающий алгоритм обратного распространения

Обратное распространение (backpropagation) является одним из ключевых алгоритмов для обучения нейронных сетей. Этот алгоритм позволяет находить оптимальные значения весов и смещений в нейронной сети, чтобы минимизировать ошибку предсказания.

Процедура обратного распространения основана на принципе градиентного спуска, который заключается в поиске локального минимума функции ошибки. В контексте нейронных сетей, функция ошибки представляет собой меру расхождения между фактическими и предсказанными значениями.

Алгоритм обратного распространения состоит из двух этапов: прямого прохода (forward pass) и обратного прохода (backward pass). Во время прямого прохода данные передаются через слои нейронной сети от входных к выходным нейронам. Каждый нейрон выполняет операцию активации, которая комбинирует взвешенные входы с помощью нелинейной функции активации.

После прямого прохода вычисляется значение функции ошибки. Затем начинается обратный проход, в котором градиент функции ошибки распространяется назад через сеть. Градиент показывает, как изменение каждого веса и смещения влияет на общую ошибку предсказания. С помощью градиента алгоритм корректирует значения весов и смещений, чтобы минимизировать ошибку.

Процедура обратного распространения требует вычисления частных производных функции ошибки по каждому параметру нейронной сети. Это делается с использованием правила цепочки (chain rule), которое позволяет разложить градиент функции ошибки на произведение множителей, соответствующих каждому слою сети.

Однако обратное распространение имеет свои ограничения. Во-первых, он может столкнуться с проблемой затухания или взрыва градиента, когда значения градиента становятся очень маленькими или очень большими. Это может замедлить или полностью остановить процесс обучения. Для решения этой проблемы используются методы нормализации градиента и выбор оптимальной скорости обучения.

Во-вторых, алгоритм обратного распространения может страдать от проблемы переобучения, когда модель слишком хорошо запоминает обучающие данные и плохо обобщает на новые примеры. Для предотвращения переобучения используются методы регуляризации, такие как добавление штрафов за сложность модели или использование методов отсечения (dropout).

Несмотря на свои ограничения, процедура обратного распространения является одним из самых эффективных алгоритмов для обучения нейронных сетей. Она позволяет находить оптимальные значения весов и смещений, чтобы минимизировать ошибку предсказания. Применение этого алгоритма в сочетании с другими техниками может значительно повысить эффективность и точность нейронных сетей.

Процедура обратного распространения является ключевой частью алгоритма обучения нейронных сетей. Она позволяет находить оптимальные значения весов и смещений, учитывая функцию ошибки и градиенты по каждому параметру. Хотя у нее есть свои ограничения, она по-прежнему является одним из наиболее эффективных методов обучения нейросетей и широко применяется в различных областях.

Сети встречного распространения как метод глубокого обучения

Сети встречного распространения (autoencoders) являются одним из наиболее популярных и эффективных методов глубокого обучения. Они представляют собой вид нейронных сетей, которые используются для автоматического выявления и представления

скрытых признаков во входных данных. Процедура обратного распространения, или backpropagation, является ключевым алгоритмом, который применяется для обучения сетей встречного распространения.

Процедура обратного распространения – это метод оптимизации, который используется для корректировки параметров нейронной сети на основе ошибки между предсказанными и ожидаемыми значениями. Она основывается на градиентном спуске – численном методе поиска минимума функции путем последовательного изменения параметров модели. Процедура обратного распространения выполняет два этапа: прямое распространение и обратное распространение ошибки.

В первом этапе – прямом распространении – данные проходят через нейросеть от входного слоя к выходному слою. Каждый нейрон в сети получает входные данные, выполняет операцию обработки и передает результат следующему нейрону. Процесс повторяется до тех пор, пока данные не достигнут выходного слоя, где происходит окончательное предсказание.

На этапе обратного распространения ошибки происходит оценка ошибки между предсказанными и ожидаемыми значениями на выходном слое. Затем эта ошибка распространяется обратно через сеть для корректировки весов и смещений каждого нейрона. Для определения величины изменения параметров сети используется градиент функции потерь по отношению к параметрам модели. Градиент – это вектор, который указывает направление наискорейшего возрастания функции.

Применение процедуры обратного распространения позволяет моделировать более сложные зависимости и выделять более информативные признаки из данных. В случае с сетями встречного распространения этот метод особенно полезен, поскольку он позволяет автоматически извлечь скрытые признаки из больших объемов данных.

Сети встречного распространения состоят из двух основных частей: энкодера и декодер. Энкодер преобразует входные данные в скрытое представление, а декодер восстанавливает данные из этого скрытого представления. В процессе обучения сети встречного распространения эти две части работают совместно для минимизации

ошибки реконструкции – разницы между исходными данными и их восстановленной версией.

Процедура обратного распространения позволяет оптимизировать параметры энкодера и декодера таким образом, чтобы минимизировать ошибку реконструкции. Благодаря этому сети встречного распространения способны автоматически выделять наиболее информативные признаки из данных.

Сети встречного распространения являются мощным инструментом глубокого обучения, который позволяет автоматически выявлять скрытые признаки во входных данных. Процедура обратного распространения играет ключевую роль в обучении таких моделей, позволяя корректировать параметры сети на основе ошибки между предсказанными и ожидаемыми значениями. Эти методы имеют широкий спектр применения в различных областях, таких как компьютерное зрение, естественный язык обработки и рекомендательные системы.

Структура сети встречного распространения

Структура сети встречного распространения (RNN) является одной из ключевых особенностей этого типа нейронных сетей. RNN состоит из повторяющихся блоков, которые позволяют информации из прошлого влиять на выводы, делаемые в настоящем. Это отличает RNN от других архитектур нейронных сетей, таких как сверточные нейронные сети или простые рекуррентные нейронные сети.

Основная идея RNN заключается в том, что каждый блок имеет свое состояние, которое передается от одного блока к другому. Это состояние содержит информацию о предыдущих шагах обработки данных и используется для принятия решений на текущем шаге. Таким образом, блоки RNN создают зависимость между данными, поступающими на вход, и результатами работы сети.

Структура RNN может быть представлена в виде графа или цикла. Каждый узел графа соответствует блоку RNN, а дуги указывают направление потока данных. Направленность графа позволяет информации перемещаться только от прошлого к будущему и запоминать предыдущие состояния. Это свойство RNN делает ее идеальным инструментом для работы с последовательными данными, такими как временные ряды или тексты.

Ключевой компонент структуры RNN – это обратное распространение ошибки. В процессе обучения нейронной сети RNN

на каждом шаге вычисляется ошибка между фактическим результатом и ожидаемым значением. Эта ошибка затем передается обратно через циклы RNN, позволяя корректировать веса и улучшать качество предсказаний.

Однако есть проблема, известная как «затухание градиента», которая может возникнуть при использовании метода обратного распространения ошибки в RNN. Затухание градиента происходит, когда градиент (производная функции потерь по весам) становится очень маленьким или исчезает на некоторых шагах времени. Это приводит к тому, что нейроны в прошлом не могут влиять на будущие выводы сети.

Для решения проблемы затухания градиента были разработаны различные модификации структуры RNN, такие как LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit). Эти модификации добавляют специальные механизмы, позволяющие сети сохранять и использовать информацию о долгосрочных зависимостях в данных.

LSTM использует специальные ячейки памяти для хранения информации из прошлого. Каждая ячейка имеет три основных компонента: входной, выходной и забывающий гейты (блоки). Входной гейт решает, какую информацию следует сохранить в ячейке памяти, а забывающий гейт контролирует степень забывания предыдущего состояния. Выходной гейт определяет, какую часть информации из ячейки следует передать на следующий шаг.

GRU также использует специальные гейты для контроля потока информации. Однако он объединяет функциональность входного и забывающего гейта в одном «обновлении» гейте. Это уменьшает сложность модели и делает ее более эффективной.

Общая структура RNN с использованием LSTM или GRU блоков может быть более сложной и содержать несколько уровней или слоев. Например, можно использовать многомерную RNN для обработки временных рядов с несколькими переменными. Каждый уровень RNN будет отвечать за обработку данных на разных временных шкалах, что позволяет моделировать сложные зависимости в данных.

Структура сети встречного распространения (RNN) является основополагающей для обработки последовательных данных и имеет специальное свойство сохранять информацию из прошлого. Обратное распространение ошибки позволяет корректировать веса сети и

улучшать ее предсказательную способность. Добавление LSTM или GRU блоков позволяет бороться с затуханием градиента и использовать долгосрочные зависимости в данных. Однако структура RNN может быть более сложной и содержать несколько уровней или слоев для обработки различных временных шкал или переменных.

Нормальное функционирование встречного распространения

Нормальное функционирование встречного распространения – одна из ключевых составляющих процесса обратного распространения в нейронных сетях. В данном подразделе мы рассмотрим, какие факторы влияют на эффективность работы сети и как их оптимизировать для достижения лучших результатов.

Одним из основных факторов, влияющих на нормальное функционирование сети, является выбор активационной функции. Активационная функция определяет выходной сигнал нейрона в зависимости от его входного значения. Встречное распространение использует различные активационные функции для каждого слоя сети. Распространенными типами активационных функций являются сигмоидная (sigmoid), гиперболический тангенс (tanh) и ReLU (Rectified Linear Unit).

Сигмоидная функция имеет форму S-образной кривой и принимает значения от 0 до 1. Она широко используется в задачах классификации, поскольку позволяет получить вероятностную интерпретацию выходных значений. Гиперболический тангенс также имеет форму S-образной кривой, но принимает значения от -1 до 1. Он используется в задачах регрессии, так как может предсказывать как положительные, так и отрицательные значения. ReLU функция является линейной на интервале [0, бесконечность] и нулевой на интервале (бесконечность, 0). Она позволяет эффективно обрабатывать большие объемы данных.

Оптимальный выбор активационной функции зависит от конкретной задачи и данных, поэтому для достижения нормального функционирования сети требуется проведение экспериментов и анализ результатов. Еще одним фактором, оказывающим влияние на работу встречного распространения – это правильная настройка гиперпараметров сети. Гиперпараметры – это параметры модели, которые не могут быть обучены на основных данных и должны быть установлены вручную. К ним относятся количество скрытых слоев,

количество нейронов в каждом слое, скорость обучения (learning rate) и момент (momentum).

Количество скрытых слоев определяет сложность модели и её способность извлекать более высокие уровни абстракции из данных. Однако слишком большое количество слоев может привести к переобучению модели, когда она становится способной хорошо обработать только тренировочные данные, но плохо справляется с новыми примерами. Количество нейронов в каждом слое также влияет на способность модели обобщать данные. Чем больше нейронов, тем более гибкая модель, однако это может привести к переобучению. Скорость обучения и момент – это параметры оптимизации процесса обратного распространения. Скорость обучения определяет шаг, с которым веса модели обновляются при каждой итерации. Высокая скорость обучения может привести к быстрой сходимости, однако может вызвать осцилляции и неустойчивость процесса. Момент – это параметр, который учитывает предыдущие изменения весов при текущем обновлении. Он помогает ускорить процесс сходимости и предотвратить застревание в локальных минимумах функции ошибки. Для достижения нормального функционирования встречного распространения необходим подходящий выбор гиперпараметров, который достигается путем экспериментов и анализа результатов.

3. ЗАДАЧА ПОСТРОЕНИЯ НЕЙРОННОЙ СЕТИ ДЛЯ ОСУЩЕСТВЛЕНИЯ ПРОГНОЗИРОВАНИЯ И РАСЧЕТА ЗНАЧЕНИЙ ПРОЦЕССА В ПРОГРАММЕ MATLAB

Для выполнения операций из таблицы 1 создайте нейронную сеть.

Рассмотрите нейронную сеть.

Проведите прогнозирование значений процесса.

Таблица 1 – Исходные данные

№	Функции	№	Функции
1	$y = \cos(x)$	4	$y = \cos(x-1)$
2	$y = \sin(2*x)$	5	$y = \sin(x-2)$
3	$y = \cos(3*x)$	6	$y = \sin(x^2)$

Пример выполнения задания:

Создадим нейронную сеть для выполнения операции y = sin (x)

X=[–1 –0.8 –0.5 –0.2 0 0.1 0.3 0.6 0.9 1];

Y =[–0.84 –0.72 –0.48 –0.19 0 0.09 0.29 0.56 0.78 0.84]

Функция nntool позволит раскрыть окно (рис. 11).

Меню «New» откроет вкладку Network/Data Manager, где выполняется аккумуляция совокупности целей и входов.

Инициируем открытие закладки Data в окне Create Network or Data и выберем поле Name, куда внесем имя переменной.

Зону Value пополним вектором значений, используя при этом кнопку Inputs размещением x, и кнопку Targets с размещением y в соответствии с рисунком 12.

Окончание процедуры ввода выполняет применением кнопки Create.

Для построения нейросети в окне Create Network or Data инициируем открытие закладки Network, как отмечно на рисунке 13.

Получаем значения x и y выбором через поля Input data и Target data соответственно.

По умолчанию будут установлены остальные параметры.

Окончание процедуры ввода выполняет применением кнопки Create.

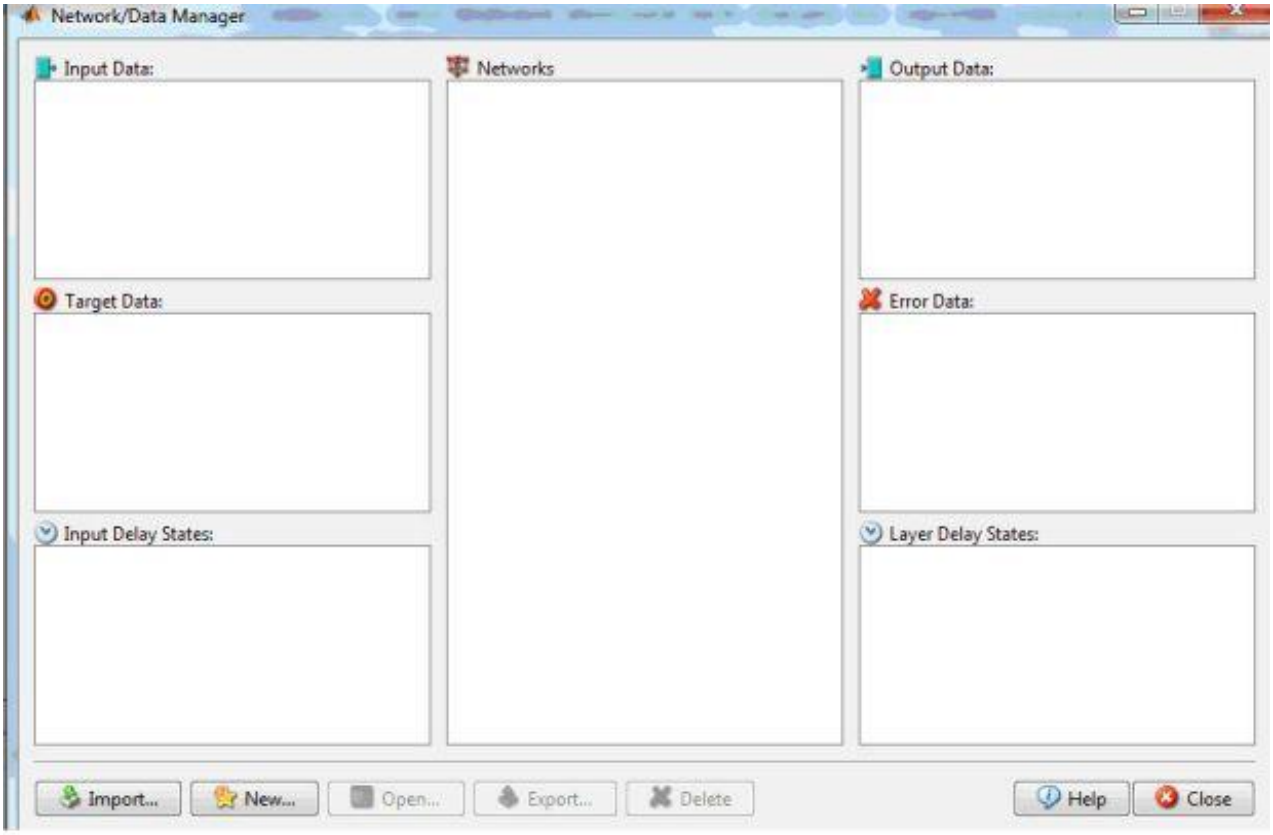

Рисунок 11 – Основное окно интерфейса

Рисунок 12 – Введение имени переменной

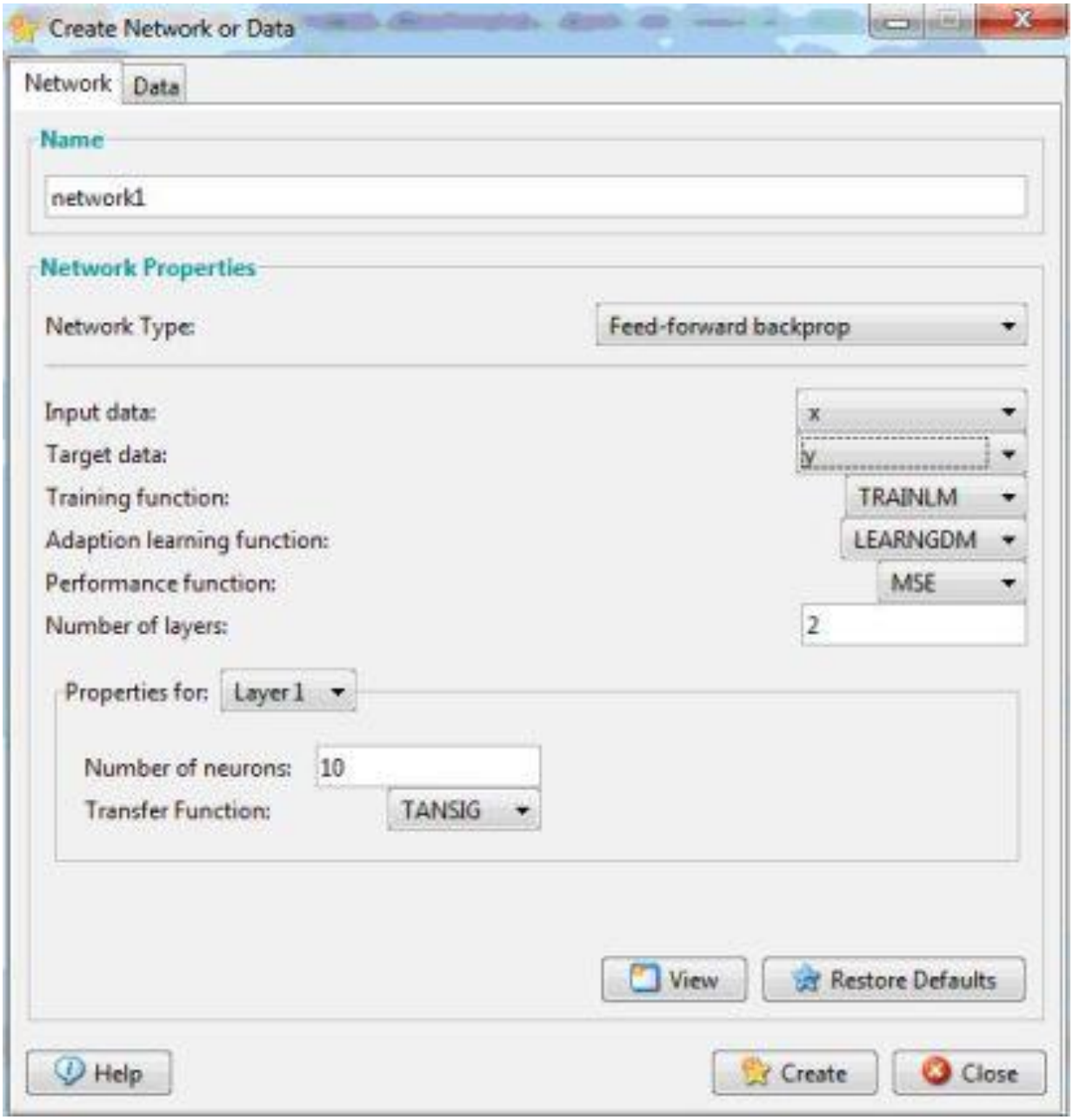

Рисунок 13 – Окно Create Network or Data

Обозначение имени созданной сети network1 зафиксируется в Network/Data Manager (рис. 14).

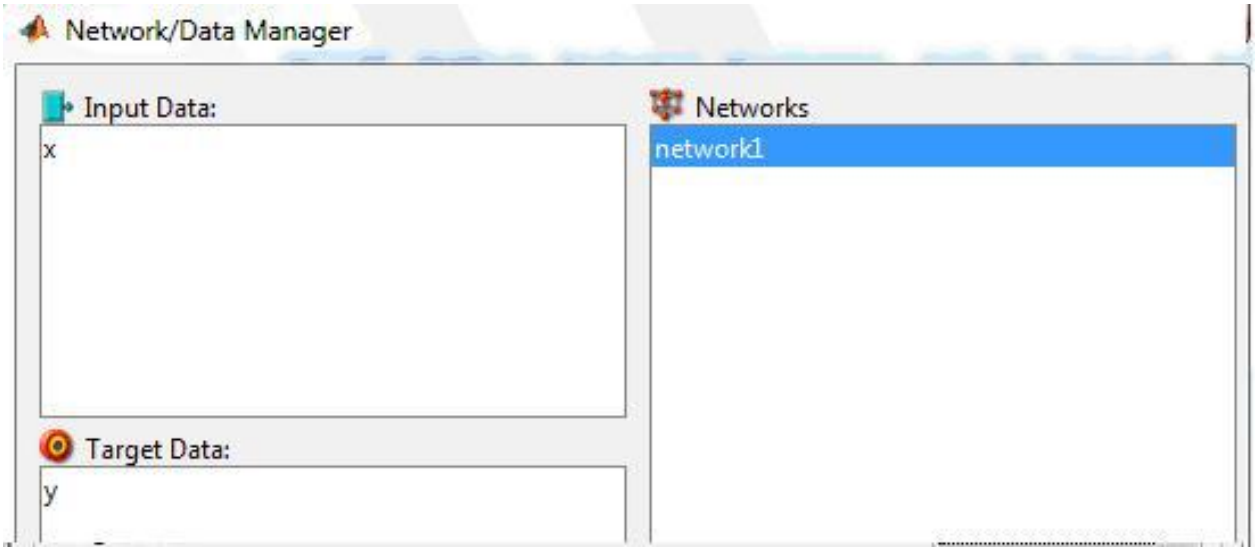

Рисунок 14 – Окно Network/Data Manager

На рисунке 15 представлена модель активизированной сети, открытой нажатием кнопки Open или кликом мыши.

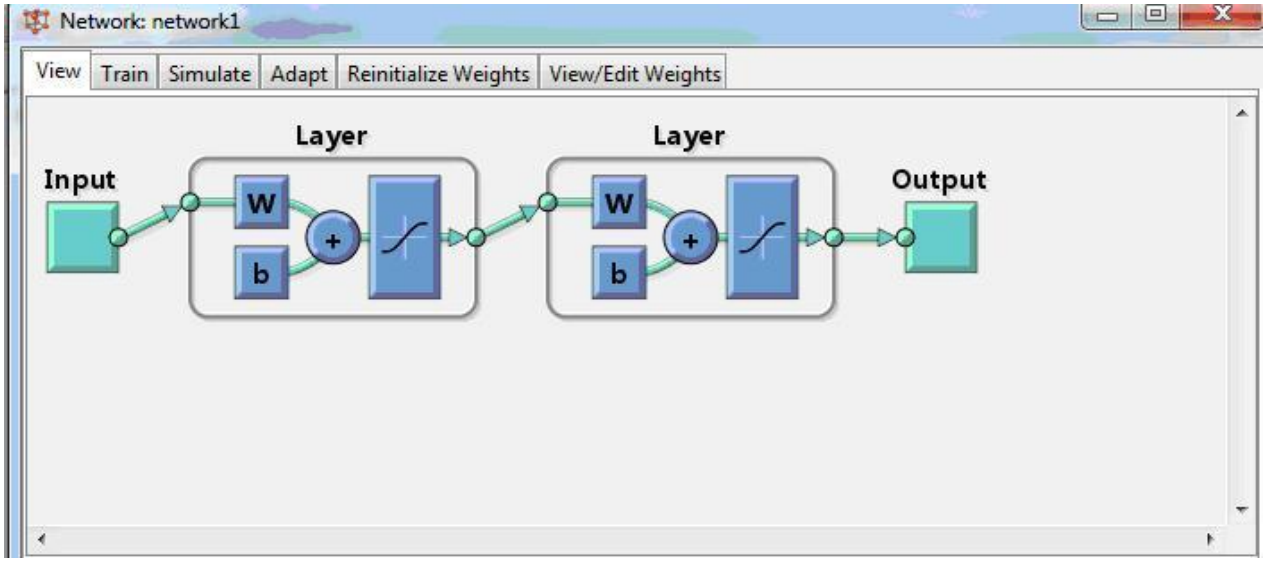

Рисунок 15 – Активизирование сети

Инициируя открытие закладок (рис. 16), имеем возможность варьирования параметрами процедуры обучения, установкой имен совокупности цели и входа, а с нажатием кнопки Train после задания всех параметров происходит обучение сети.

Результаты окончания обучения можно увидеть в окне, как показано на рисунке 16.

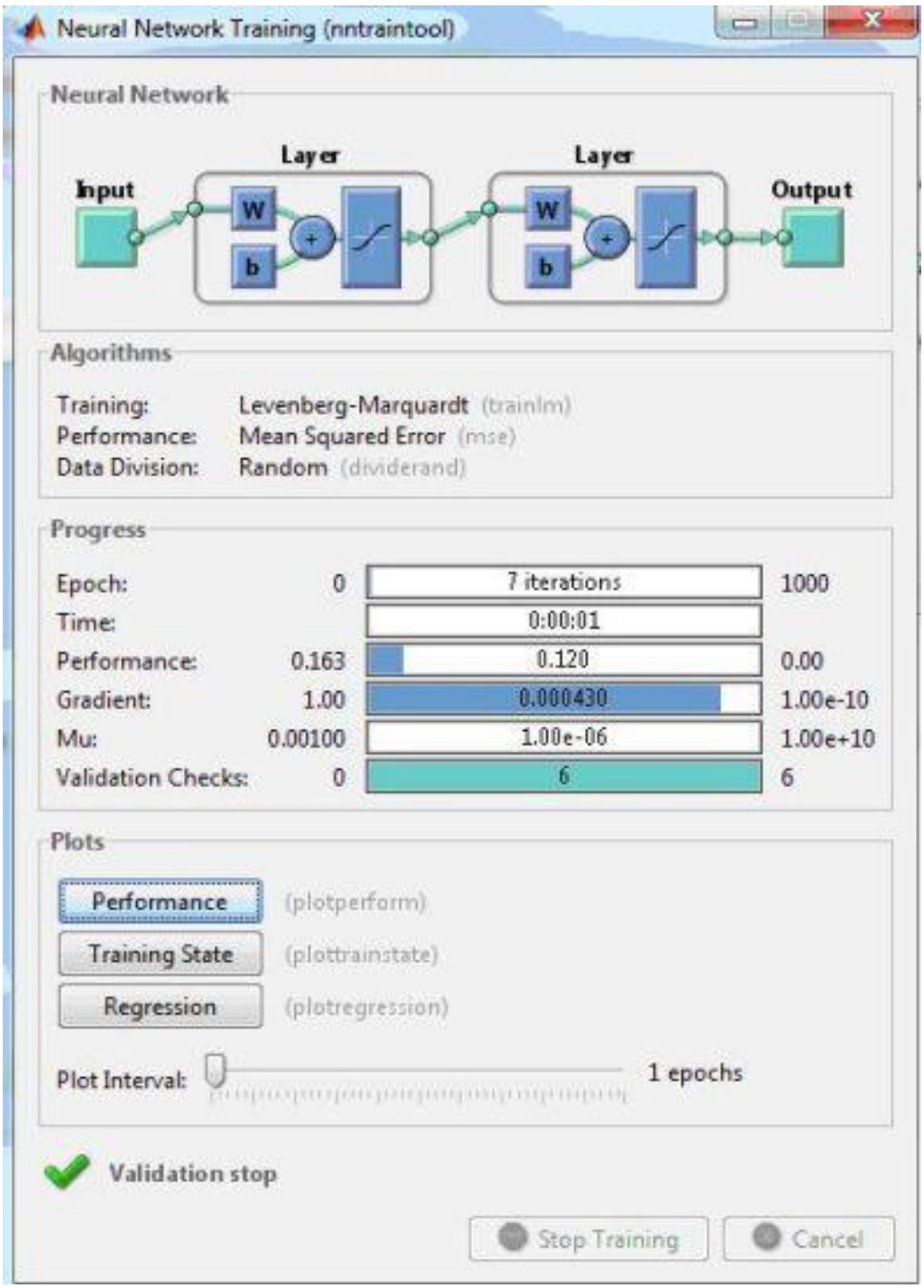

Рисунок 16 – Результаты обучения

Графические результаты обучения нейросети можно увидеть нажатием кнопки Performance, как показано на рисунке 17.

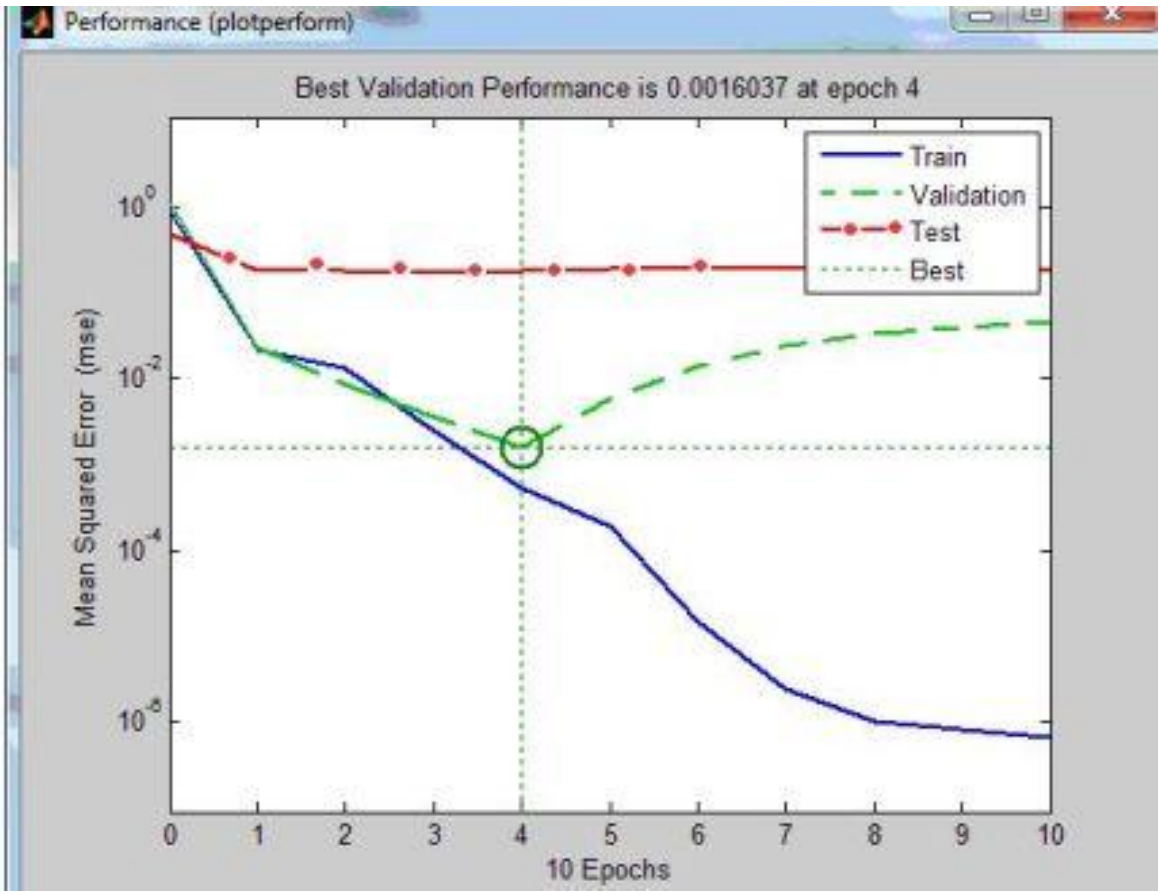

Рисунок 17 – Результаты обучения в графическом виде

Высокая точность аппроксимации определена в абсолютной погрешности 0,0366, относительной погрешности 3,66%.

Значения ошибок размещены в окне Data: network1_errors (рис. 18).

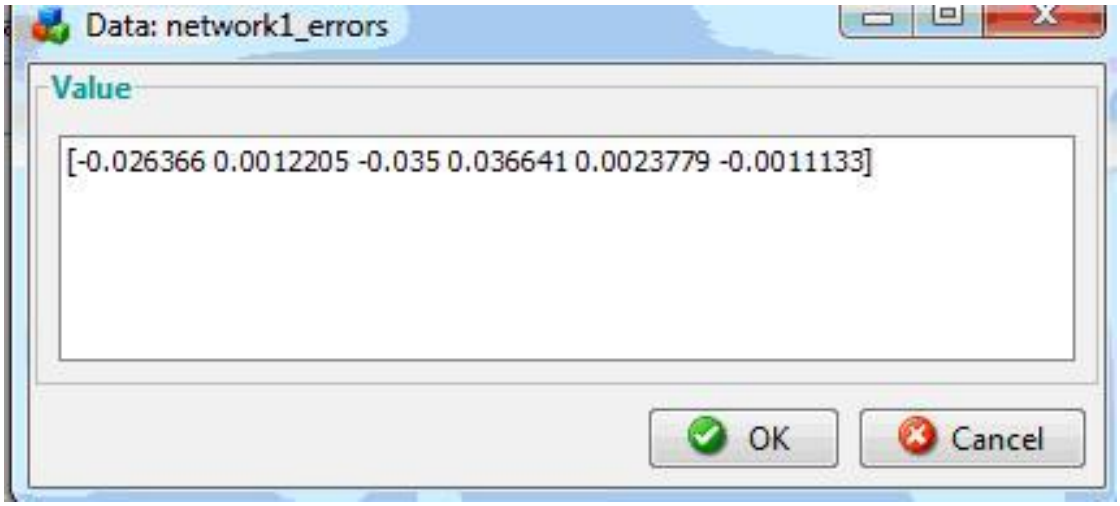

Рисунок 18 – Окно Data: network1_errors

Кроме того, построение нейросети возможно в среде Simulink, где выполнится ее схематичное изображение, как показано на рисунке 19.

Инициируем функцию gensim (network1) введением в командное окно.

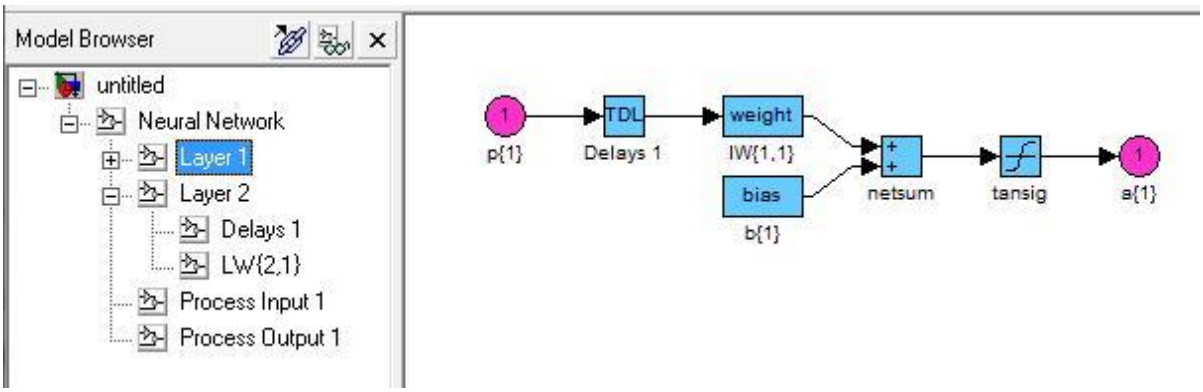

Рисунок 19 – Схема

Выполненная схема отображает ее функциональность в реализации процесса моделирования нейронной сети.

Рассмотрим образец прогнозирования функции.

Программа создания сети с помощью функции newlind без дополнительного обучения:

```
x=0:0.25:5;
% Задание диапазона времени от нуля до пять секунд
y=sin(x);% предсказываемый сигнал
>> Q=length(y); % Определение количества точек вектор
>> P=zeros(5, Q); % Создание нулевой матрицы P
>> % Создание входных векторов в виде строк матрицы
>> P(1,2:Q)=y(1,1:(Q-1));
>> P(2,3:Q)=y(1,1:(Q-2);
>> P(2,3:Q)=y(1,1:(Q-2));
>> P(3,4:Q)=y(1,1:(Q-3));
>> P(4,5:Q)=y(1,1:(Q-4));
>> P(5,6:Q)=y(1,1:(Q-5));
>> s=newlind(P,y=); % Создание новой НС с именем s
>> z=sim(s, P); % Расчет прогнозируемых значений
>> % Создание графиков исходного сигнала и прогноза
>> plot(x,z, x, y=,'*')
>> ylabel(' Прогнозируемые и исходный сигналы')
>> xlabel('Время')
```

На рисунке 20 представлен график результата обучения.

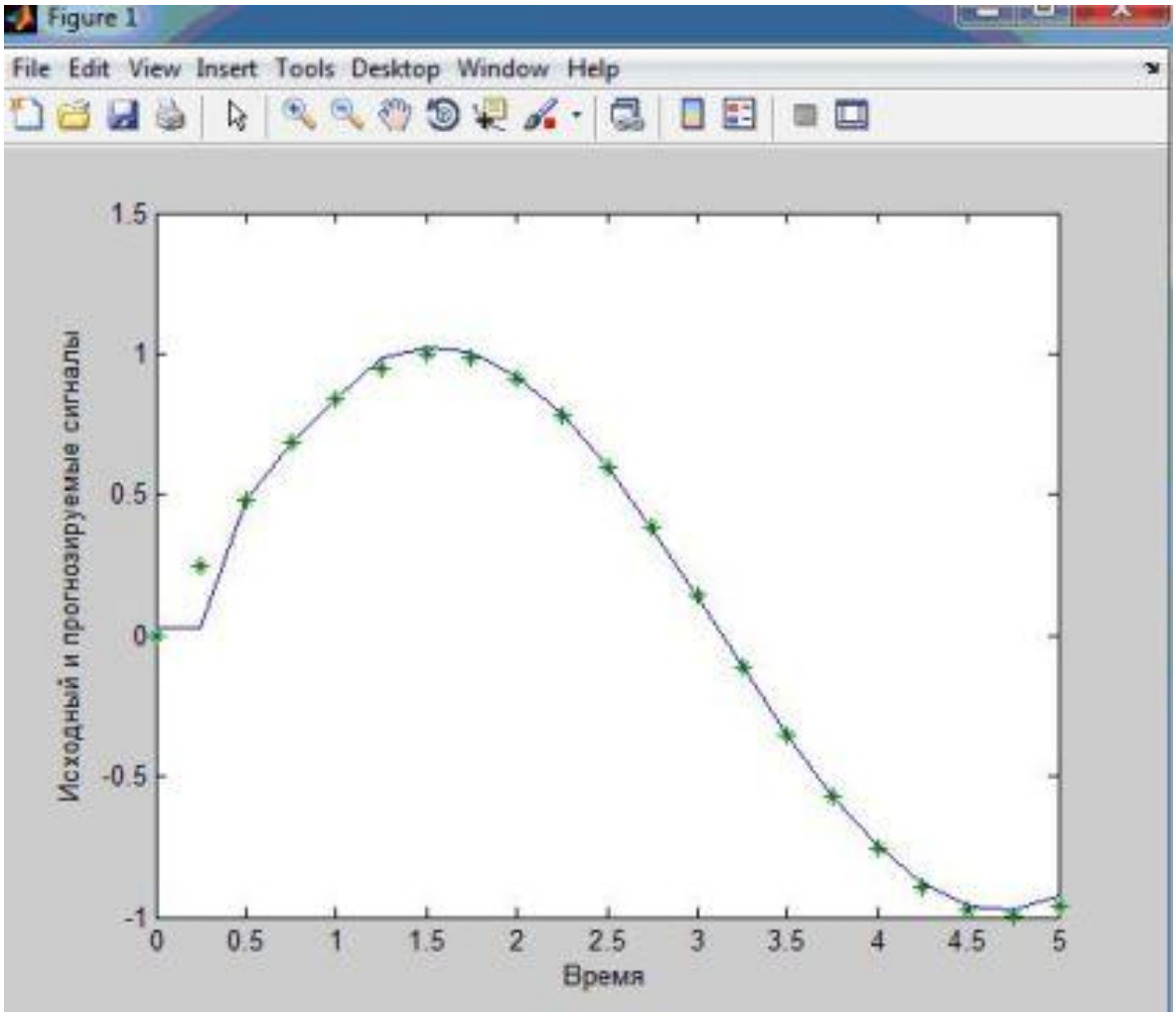

Рисунок 20 – График результата

4. ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ ПРОГРАММИРОВАНИЯ Python

Python – это высокоуровневый язык программирования, который отличается своей простотой и интуитивно понятным синтаксисом. Созданный в конце 1980-х годов Гвидо ван Россумом, Python приобрел значительную популярность в IT-сообществе благодаря своей гибкости и эффективности.

Гвидо ван Россум разработал Python как язык программирования, который был прост в использовании и читаем в написании кода.

Один из главных преимуществ Python – его удобство при чтении и написании кода. В отличие от многих других языков программирования, Python использует простой и понятный английский язык, который делает его доступным даже для новичков. Это позволяет разработчикам быстро овладеть языком и сократить время на программирование.

Python – это также интерпретируемый язык программирования, что означает, что код Python не нуждается в компиляции перед запуском. Это упрощает и ускоряет процесс разработки, так как программисты могут немедленно проверять результаты своей работы без необходимости ждать компиляции.

Python также известен своим богатым набором встроенных библиотек и модулей, которые позволяют разработчикам выполнять широкий спектр задач. С этими библиотеками можно работать с базами данных, создавать графические интерфейсы, обрабатывать изображения и многое другое. Вот несколько популярных библиотек Python:

– NumPy: Библиотека для работы с многомерными массивами и математическими функциями. Она широко используется для научных вычислений, обработки данных и машинного обучения.

– pandas: Библиотека для работы с табличными данными. Она предоставляет удобные структуры данных и функции для обработки, анализа и визуализации данных.

– matplotlib: Библиотека для создания графиков и визуализации данных. С помощью matplotlib вы можете создавать различные типы графиков, диаграмм и даже анимаций.

– TensorFlow: Библиотека для машинного обучения и глубокого обучения. Она предоставляет инструменты для создания и обучения нейронных сетей и моделей машинного обучения.

– Scikit-learn: Библиотека для машинного обучения, которая содержит реализацию различных алгоритмов классификации, регрессии, кластеризации и многих других.

– Flask: Легковесный фреймворк для создания веб-приложений. Он позволяет создавать веб-серверы, обрабатывать запросы и отображать HTML-страницы.

– Django: Полнофункциональный веб-фреймворк, предназначенный для создания сложных веб-приложений. Django имеет мощные инструменты для работы с базами данных, аутентификации пользователей, административной панели и многое другое.

Это только небольшая часть популярных библиотек Python. В зависимости от конкретной задачи, можно найти множество других библиотек, которые помогут в разработке.

Python – это мощный язык программирования, который поставляется вместе с обширной библиотекой стандартных модулей. Стандартные модули Python предоставляют различные функции и инструменты, которые упрощают разработку приложений и решение различных задач. Например:

– Модуль datetime

Модуль datetime предоставляет классы для работы с датами и временем. Он позволяет выполнять различные операции с датами, такие как создание даты, получение текущей даты и времени, вычисление разницы между двумя датами и многое другое.

python import datetime

current_date datetime.date.today() print(current_date) #

Выводит текущую дату

current_time datetime.datetime.now() print(current_time) #

Выводит текущую дату и время

date_1 datetime.date(2021, 1, 1) date_2 datetime.date(2022, 1, 1)

difference date_2 - date_1 print(difference.days) #

Выводит разницу в днях

– Модуль math

Модуль math предоставляет функции для выполнения математических операций. Он содержит функции для вычисления тригонометрических функций, логарифмов, экспоненты, округления чисел и многое другое.

python import math

result math.sqrt(25) print(result) #

Выводит 5.0
result math.sin(math.pi / 2) print(result) #
Выводит 1.0
result math.log(10) print(result) #
Выводит 2.302585092994046
– Модуль json

Модуль json предоставляет функции для работы с JSON (JavaScript Object Notation). Он позволяет преобразовывать данные из формата JSON в формат Python и наоборот, а также выполнять различные операции с данными в формате JSON.

python import json
data {"name": "John", "age": 30}
json_data json.dumps(data) print(json_data) #
Выводит {"name": "John", "age": 30}
python_data json.loads(json_data) print(python_data) #
Выводит {'name': 'John', '

Большое сообщество разработчиков Python также активно разрабатывает и делится своими модулями и пакетами, что помогает ускорить процесс разработки и сделать его более эффективным.

Еще одним преимуществом языка Python является его кросс-платформенность. Это означает, что код, написанный на Python, может быть запущен на различных операционных системах, таких как Windows, macOS и Linux. Эта возможность делает Python идеальным выбором для разработки приложений, которые должны быть доступны на разных платформах.

В целом, Python представляет собой мощный и гибкий инструмент для разработки приложений. Благодаря своей простоте, кросс-платформенности и обширным возможностям, Python позволяет разработчикам эффективно создавать различные программные решения, от простых сценариев до сложных приложений. Будь то веб-разработка, анализ данных, машинное обучение или другие области, Python прекрасно подходит для любого проекта.

5. ИСПОЛЬЗОВАНИЕ БИБЛИОТЕК Python ДЛЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА И МАШИННОГО ОБУЧЕНИЯ

Язык программирования Python использует ряд библиотек для искусственного интеллекта и машинного обучения. Они широко используются в промышленности и зарекомендовали себя как мощные инструменты для построения моделей искусственного интеллекта. Рассмотрим некоторые из них.

«TensorFlow – это библиотека с открытым исходным кодом, разработанная Google, для создания и развёртывания моделей машинного обучения. Это одна из самых популярных библиотек для искусственного интеллекта и машинного обучения, которая используется такими компаниями, как Airbnb, Intel и др. TensorFlow отлично подходит для построения нейронных сетей и моделей глубокого обучения, а также обладает широким спектром инструментов для построения и обучения моделей.

Как использовать TensorFlow для построения простой нейронной сети» [37]:

```
import tensorflow as tf

# define the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, input_shape=(8,), activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

«Scikit-learn – это широко используемая библиотека для машинного обучения в Python. Она построена поверх NumPy и SciPy и предлагает широкий спектр инструментов для создания и оценки моделей машинного обучения. Scikit-learn отлично подходит для построения традиционных моделей машинного обучения, таких как

линейная регрессия, деревья решений и кластеризация k-средних» [37].

«Как использовать scikit-learn, чтобы построить простую модель линейной регрессии» [37]:

```
from sklearn.linear_model import LinearRegression

# create the model
model = LinearRegression()

# fit the model to the data
model.fit(X_train, y_train)

# make predictions
y_pred = model.predict(X_test)
```

«Keras – это высокоуровневая библиотека нейронных сетей для Python. Она создана поверх TensorFlow и предназначена для того, чтобы максимально упростить построение и обучение нейронных сетей. Keras отлично подходит для построения моделей глубокого обучения и обладает широким спектром инструментов для построения и обучения моделей» [37].

«Как использовать Keras для построения простой нейронной сети» [37]:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# define the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

«Pandas – это библиотека для обработки и анализа данных на Python. Она широко используется для работы со структурированными данными и отлично подходит для очистки, преобразования и анализа данных. Pandas имеет широкий спектр инструментов для работы с данными, включая объекты dataframe и series, которые похожи на таблицы и столбцы в SQL» [37].

«Как использовать Pandas для загрузки и изучения набора данных» [37]:

```
import pandas as pd

# load the data
data = pd.read_csv('data.csv')

# explore the data
print(data.head())
print(data.describe())
```

«NumPy – это библиотека для численных вычислений на Python. Она широко используется для работы с массивами и матрицами и отлично подходит для выполнения математических операций с данными. NumPy часто используется в сочетании с другими библиотеками, такими как SciPy и Pandas, для обработки и анализа данных» [37].

«Как использовать NumPy для создания массивов и управления ими» [37]:

```
import numpy as np

# create an array
a = np.array([1, 2, 3, 4])

# perform mathematical operations on the array
b = a * 2
c = a + b

# index and slice the array
print(a[2])
print(b[1:3])
```

«Matplotlib – это библиотека для визуализации данных на Python. Она широко используется для создания графиков и диаграмм, а также отлично подходит для визуализации данных. Matplotlib обладает широким спектром инструментов для создания различных типов графиков и часто используется в сочетании с другими библиотеками, такими как Pandas, для исследования данных» [37].

«Как использовать Matplotlib для создания простого точечного графика» [37]:

```
import matplotlib.pyplot as plt

# create some data
x = [1, 2, 3, 4]
y = [2, 4, 6, 8]

# create the scatter plot
plt.scatter(x, y)

# add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot')

# show the plot
plt.show()
```

«Seaborn – это библиотека для визуализации данных на Python. Она построен поверх Matplotlib и предназначена для того, чтобы максимально упростить создание красивых и информативных графиков. Seaborn отлично подходит для создания статистических графиков и часто используется в сочетании с другими библиотеками, такими как Pandas и NumPy, для исследования данных» [37].

«Как использовать Seaborn для создания простого штрихового графика» [37]:

```
import seaborn as sns

# create some data
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}

# create the bar plot
sns.barplot(data=data)

# add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Bar Plot')

# show the plot
plt.show()
```

«NLTK (Natural Language Toolkit) – это библиотека для обработки естественного языка в Python. Она широко используется для работы с текстовыми данными и отлично подходит для таких задач, как классификация текста, анализ отношений и языковой перевод. NLTK обладает широким спектром инструментов для работы с текстовыми данными, включая токенизацию, стемминг и лемматизацию» [37].

«Как использовать NLTK для обозначения предложения» [37]:

```
import nltk

# download the necessary resources
nltk.download('punkt')

# tokenize a sentence
sentence = "This is a sentence."
tokens = nltk.word_tokenize(sentence)
print(tokens)
```

«Gensim – это библиотека для неконтролируемого тематического моделирования и анализа сходства документов на Python. Она широко используется для таких задач, как обобщение

текста, кластеризация документов и тематическое моделирование. Gensim обладает широким спектром инструментов для работы с текстовыми данными, включая word2vec и LDA (скрытое распределение Дирихле)» [37].

«Как использовать Gensim для обучения модели word2vec» [37]:

```
from gensim.models import Word2Vec

# create a list of sentences
sentences = [['This', 'is', 'sentence', 'one'], ['This', 'is', 'sentence', 'two']]

# train the model
model = Word2Vec(sentences, min_count=1)

# print the results
print(model.wv['sentence'])
```

«OpenCV – это библиотека для компьютерного зрения на Python. Она широко используется для таких задач, как обработка изображений и видео, обнаружение объектов и распознавание лиц. OpenCV обладает широким спектром инструментов для работы с изображениями и видео, включая фильтрацию изображений, обнаружение объектов и извлечение объектов» [37].

«Как использовать OpenCV для загрузки и отображения изображения» [37]:

```
import cv2

# load the image
image = cv2.imread('image.jpg')

# display the image
cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

«Это были десять лучших библиотек Python для искусственного интеллекта и машинного обучения. Они широко используются в промышленности и зарекомендовали себя как мощные инструменты для построения моделей искусственного интеллекта. Независимо от того, создаёте ли вы нейронную сеть, модель глубокого обучения или традиционную модель машинного обучения, в этих библиотеках есть инструменты, необходимые для выполнения работы» [37].

«Эти библиотеки не ограничиваются приведёнными здесь примерами, они предлагают гораздо больше функциональных возможностей. Лучший способ получить представление об их полных возможностях – это изучить их документацию и поэкспериментировать с ними в своих собственных проектах.

Имейте в виду, что эти библиотеки постоянно развиваются, регулярно выпускаются новые функции и обновления. Важно быть в курсе последних разработок и пользоваться преимуществами новых функций по мере их появления.

Стоит отметить, что эти библиотеки не единственные, доступные для искусственного интеллекта и машинного обучения в Python. Существует множество других замечательных библиотек, таких как PyTorch, LightGBM и Scipy, которые также стоит изучить.

В целом, Python – отличный выбор для искусственного интеллекта и машинного обучения. С помощью этих мощных библиотек легко создавать и развёртывать модели, которые могут решать реальные проблемы. Независимо от того, являетесь ли вы новичком или опытным разработчиком, эти библиотеки предоставляют инструменты, необходимые для того, чтобы вывести ваши проекты в области искусственного интеллекта и машинного обучения на новый уровень» [37].

Рассмотрим подробнее некоторые библиотеки.

TensorFlow

«Установка и импорт TensorFlow

Чтобы начать использовать TensorFlow в Python, нужно установить эту библиотеку. Это можно сделать с помощью менеджера пакетов pip:

```
pip install tensorflow
```

После этого TensorFlow можно импортировать в коде Python:

```
import tensorflow as tf
```

TensorFlow имеет высокоуровневый API (tf.keras) и низкоуровневый API. В данной статье мы будем использовать низкоуровневый API для большего понимания принципов работы» [15].

«Импортируем основные компоненты:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Теперь TensorFlow доступен для использования в нашем коде на Python для создания и обучения нейронных сетей.

Создание и обучение простой нейронной сети

Давайте создадим и обучим простую полно-связную нейронную сеть на TensorFlow для решения задачи классификации.

Сначала определяем архитектуру – последовательность слоев» [15]. «Создадим 3 полно-связных слоя с 64, 32 и 10 нейронами:

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10))
```

Для обучения скомпилируем модель с параметрами: функция потерь, оптимизатор и метрики:

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Далее обучим сеть на тренировочных данных:

```
model.fit(train_images, train_labels, epochs=5)
```

После обучения сеть можно использовать для классификации новых изображений» [15].

«Использование различных слоев и функций

TensorFlow предоставляет большой набор готовых слоев и функций для построения нейронных сетей:

– Полносвязные слои (Dense) – основа для создания полносвязных сетей;

– Сверточные слои (Conv2D, Conv3D) – используются в сверточных сетях для работы с изображениями;

– Пулинг слои (MaxPooling2D) – применяются после сверточных для снижения размерности;

– Рекуррентные слои (LSTM, GRU) – для сетей RNN и обработки последовательностей;

– Слои нормализации (BatchNormalization) – для нормализации активаций в сети;

– Функции активации (ReLU, LeakyReLU, Sigmoid, Softmax и др.) - добавляют нелинейность;

– Функции потерь (Losses) – MSE, CrossEntropy, SparseCategoricalCrossentropy и др.;

– Оптимизаторы (Optimizers) – Adam, SGD, RMSprop и др. для обновления весов.

Комбинируя разные слои, можно создавать нейросети практически любой архитектуры для решения широкого круга задач» [15].

«Визуализация работы нейросети в TensorFlow

Чтобы лучше понимать и отлаживать нейронные сети, очень полезно визуализировать их работу. В TensorFlow есть инструменты для визуализации:

TensorBoard – позволяет строить графики потерь, метрик, весов нейронов в процессе обучения:

```
tensorboard = TensorBoard(log_dir="logs")

model.fit(data, labels, epochs=10, callbacks=[tensorboard])
```

TensorBoard запускается в браузере и отображает графики из логов обучения.

Кроме того, можно визуализировать активации конкретных нейронов при обработке входных данных с помощью Keras:

```
layer = model.layers[2]
activations = layer.activations

import matplotlib.pyplot as plt
plt.imshow(activations[0][0,:,:], cmap='viridis')
```

Это помогает понимать, на что именно реагируют нейроны в разных слоях сети.

Визуализация значительно упрощает отладку и оптимизацию нейронных сетей в TensorFlow» [15].

Keras

«Установка и импорт Keras

Keras – это высокоуровневая нейро-сетевая библиотека для Python, которая может использовать TensorFlow в качестве бэкенда. Установить Keras можно через pip:

```
pip install keras
```

Импортировать:

```
import keras
from keras import models
from keras import layers
```

Keras имеет простой и понятный API для быстрой разработки нейронных сетей.

В Keras есть два основных способа создания моделей*:*

– Последовательный (Sequential) – для линейных стеков слоев;

– Функциональный (Functional) – для произвольных графов слоев» [15].

«Рассмотрим использование каждого из них.

Создание модели Sequential в Keras

Чтобы создать простую линейную модель в Keras, используется API Sequential:

```
model = keras.Sequential()

model.add(keras.layers.Dense(32, activation='relu'))
model.add(keras.layers.Dense(10, activation='softmax'))
```

Слои просто добавляются в модель последовательно один за другим с помощью метода add(). Это удобный способ быстро создать полносвязную или сверточную сеть.

Далее модель компилируется и обучается стандартными методами:

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

model.fit(x_train, y_train, batch_size=64, epochs=5)
```

Таким образом, с помощью простого Sequential API в Keras можно определять и обучать нейронные сети всего несколькими строками кода» [15].

«Создание модели Functional в Keras

Для создания более сложных сетевых архитектур в Keras используется функциональный API:

```
inputs = keras.Input(shape=(32,))

x = layers.Dense(64, activation='relu')(inputs)
x = layers.Dense(64, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

Здесь мы определяем отдельные слои, а затем связываем их в единую модель, указывая входные и выходные тензоры.

Это позволяет создавать разветвленные сети, остаточные связи, многовходовые и многовыходные модели.

Например, можно соединить две разные модели Sequential в одну большую модель Functional.

Такой API дает гибкость в создании нейро-сетевых архитектур произвольной сложности в Keras» [15].

«Преимущества Keras как «обертки» над TensorFlow

Keras имеет ряд преимуществ по сравнению с TensorFlow:

– Более простой и интуитивный API для разработки моделей;

– Меньше кода для решения типовых задач;

– Встроенные функции для обучения, оптимизации, оценки моделей;

– Возможность быстрого прототипирования и итераций;

– Совместимость с TensorFlow как низкоуровневым бэкендом.

Но при этом Keras сохраняет гибкость TensorFlow для разработки сложных архитектур.

Примеры обучения разных типов нейросетей в Keras:

– Полносвязные для классификации текста или чисел;

– Сверточные для обработки изображений;

– Рекуррентные (LSTM, GRU) для текста и временных рядов;

– Автокодировщики для снижения размерности;

– Сети на основе GAN для генерации контента.

Keras позволяет легко реализовывать, обучать и применять эти типы архитектур для решения практических задач» [15].

«Решение задачи классификации изображений с TensorFlow

Рассмотрим применение TensorFlow для решения классической задачи компьютерного зрения – классификации изображений.

В качестве данных возьмем один из популярных датасетов – MNIST, содержащий 70 000 изображений рукописных цифр 0-9.

Построим простую сверточную нейронную сеть в TensorFlow:

– Слой свертки для извлечения признаков;

– Пулинг слой для снижения размерности;

– Полносвязный слой для классификации.

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

Обучим модель на данных MNIST:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

Теперь сеть готова классифицировать рукописные цифры с достаточно высокой точностью.

Классификация изображений – один из самых распространенных примеров применения нейронных сетей на практике с помощью TensorFlow» [15].

Анализ временных рядов с помощью RNN

«Рекуррентные нейронные сети (RNN) хорошо подходят для анализа последовательных данных, таких как временные ряды.

Рассмотрим применение RNN на TensorFlow для прогнозирования временного ряда.

Построим простую RNN-сеть с использованием lstm-слоев:

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(128, input_shape=(None, 1)))
model.add(tf.keras.layers.Dense(1))

model.compile(loss='mae', optimizer='adam')
```

Обучим сеть на данных о ценах акций за несколько лет для прогноза цены через месяц:

```
model.fit(X_train, y_train, epochs=10)
```

Теперь используем обученную RNN для предсказания цены на месяц на основе предыдущих данных:

```
prediction = model.predict(X_test)
```

LSTM-сети отлично подходят для многих задач анализа временных рядов: прогнозирование, выявление аномалий, классификация и др.

Мы рассмотрели основные возможности библиотеки TensorFlow для создания и обучения нейронных сетей с помощью Python» [15].

6. СПОСОБЫ РЕАЛИЗАЦИИ НЕЙРОННЫХ СЕТЕЙ В Python

Далее будет представлено объяснение того, как работают нейронные сети, а также показаны способы их реализации в Python.

Создание нейронных блоков

Для начала необходимо определиться с тем, что из себя представляют базовые компоненты нейронной сети – нейроны. Нейрон принимает вводные данные, выполняет с ними определенные математические операции, а затем выводит результат. Нейрон с двумя входными данными выглядит следующим образом:

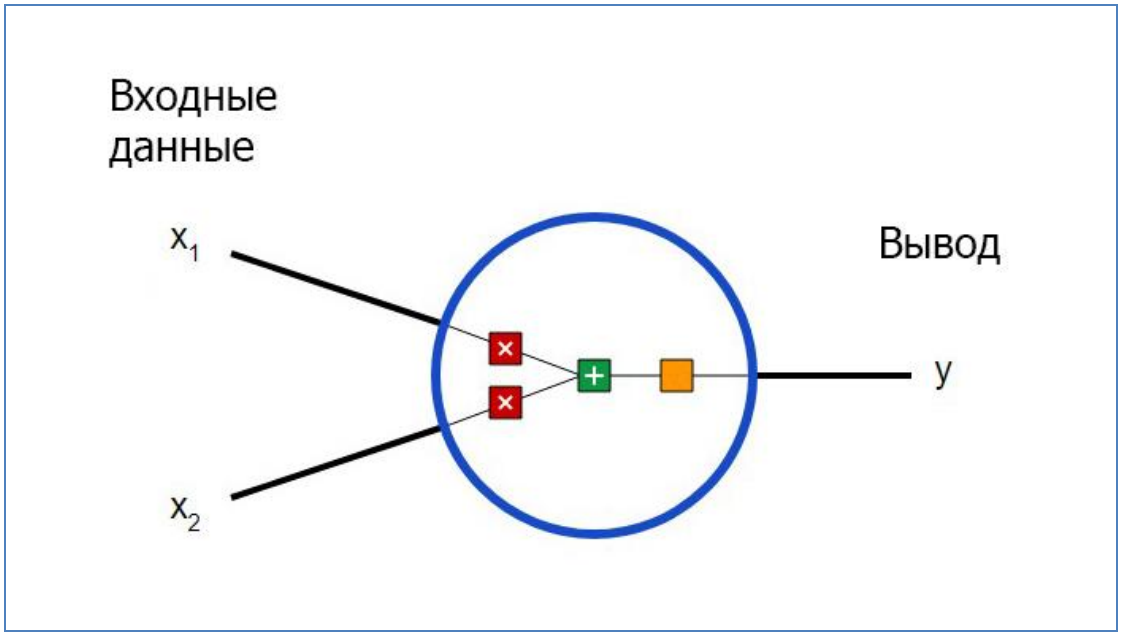

Рисунок 21 – Базовые компоненты нейронной сети

Здесь происходят три вещи. Во-первых, каждый вход умножается на вес (рис. 21):

$$x_1 \rightarrow x_1 * w_1$$
$$x_2 \rightarrow x_2 * w_2$$

Затем все взвешенные входы складываются вместе со смещением b (рис. 22):

$$(x_1 * w_1) + (x_2 * w_2) + b$$

Наконец, сумма передается через функцию активации (рис. 22):

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Функция активации используется для подключения несвязанных входных данных с выводом, у которого простая и предсказуемая форма. Как правило, в качестве используемой функцией активации берется функция сигмоида:

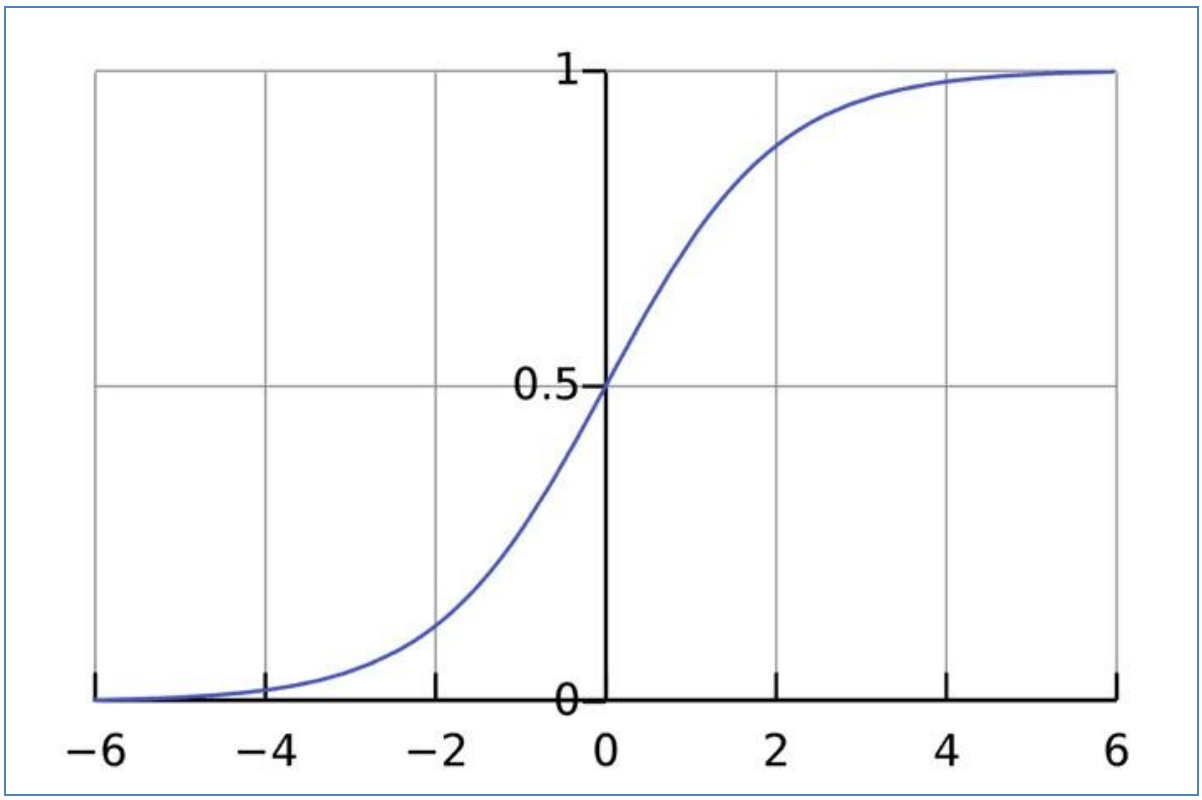

Рисунок 22 – Функция сигмоида

Функция сигмоида (рис. 22) выводит только числа в диапазоне (0, 1). Вы можете воспринимать это как компрессию от (−∞, +∞) до (0, 1). Крупные отрицательные числа становятся ~0, а крупные положительные числа становятся ~1.

Простой пример работы с нейронами в Python

Предположим, у нас есть нейрон с двумя входами, который использует функцию активации сигмоида и имеет следующие параметры:

$$w = [0, 1]$$
$$b = 4$$

$w = [0,1]$ – это просто один из способов написания $w_1 = 0$, $w_2 = 1$ в векторной форме. Присвоим нейрону вход со значением $x = [2, 3]$. Для более компактного представления будет использовано скалярное произведение.

$$(w \cdot x) + b = \big((w_1 * x_1) + (w_2 * x_2)\big) + b = 0 * 2 + 1 * 3 + 4 = 7$$
$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

С учетом, что вход был $x = [2, 3]$, вывод будет равен 0.999. Вот и все. Такой процесс передачи входных данных для получения вывода называется прямым распространением, или feedforward.

Создание нейрона с нуля в Python

Приступим к имплементации нейрона. Для этого потребуется использовать NumPy. Это мощная вычислительная библиотека Python, которая задействует математические операции:

Python

```
import numpy as np


def sigmoid(x):
    # Наша функция активации: f(x) = 1 / (1 + e^(-x))
    return 1 / (1 + np.exp(-x))


class Neuron:
    def __init__(self, weights, bias):
        self.weights = weights
        self.bias = bias

    def feedforward(self, inputs):
        # Вводные данные о весе, добавление смещения
        # и последующее использование функции активации

        total = np.dot(self.weights, inputs) + self.bias
        return sigmoid(total)


weights = np.array([0, 1])  # w1 = 0, w2 = 1
bias = 4  # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])  # x1 = 2, x2 = 3
print(n.feedforward(x))  # 0.9990889488055994
```

Узнаете числа? Это тот же пример, который рассматривался ранее. Ответ полученный на этот раз также равен 0.999.

Пример сбора нейронов в нейросеть

Нейронная сеть по сути представляет собой группу связанных между собой нейронов. Простая нейронная сеть выглядит следующим образом (рис. 23):

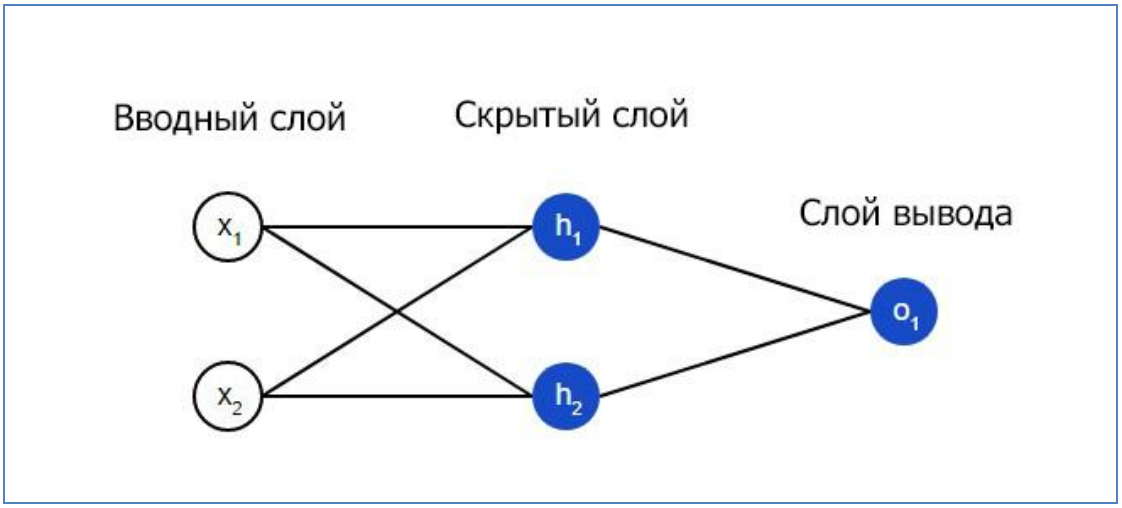

Рисунок 23 – Простая нейронная сеть

На вводном слое сети два входа – x_1 и x_2. На скрытом слое два нейтрона – h_1 и h_2. На слое вывода находится один нейрон – o_1. Обратите внимание на то, что входные данные для o_1 являются результатами вывода h_1 и h_2. Таким образом и строится нейросеть.

Скрытым слоем называется любой слой между вводным слоем и слоем вывода, что являются первым и последним слоями соответственно. Скрытых слоев может быть несколько.

Пример прямого распространения FeedForward

Давайте используем продемонстрированную выше сеть и представим, что все нейроны имеют одинаковый вес $w = [0, 1]$, одинаковое смещение $b = 0$ и ту же самую функцию активации сигмоида. Пусть h_1, h_2 и o_1 сами отметят результаты вывода представленных ими нейронов.

Что случится, если в качестве ввода будет использовано значение $x = [2, 3]$?

$$h_1 = h_2 = f(w \cdot x + b) = f\big((0 * 2) + (1 * 3) + 0\big) = f(3) = \\ = \boxed{0.9526}$$

$$o_1 = f(w \cdot [h_1, h_2] + b) = f\big((0 * h_1) + (1 * h_2) + 0\big) = f(0.9526) = \\ = \boxed{0.7216}$$

Результат вывода нейронной сети для входного значения x = [2, 3] составляет 0.7216.

Нейронная сеть может иметь любое количество слоев с любым количеством нейронов в этих слоях.

Суть остается той же: нужно направить входные данные через нейроны в сеть для получения в итоге выходных данных. Для простоты далее будет создан код сети.

Создание нейронной сети прямого распространения FeedForward

Далее будет показано, как реализовать прямое распространение feedforward в отношении нейронной сети. В качестве опорной точки будет использована следующая схема нейронной сети (рис. 24):

Python

```
import numpy as np

# ... Здесь код из предыдущего раздела


class OurNeuralNetwork:
  """
  Нейронная сеть, у которой:
    - 2 входа
    - 1 скрытый слой с двумя нейронами (h1, h2)
    - слой вывода с одним нейроном (o1)
  У каждого нейрона одинаковые вес и смещение:
    - w = [0, 1]
    - b = 0
  """
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    # Класс Neuron из предыдущего раздела
    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)

    # Вводы для o1 являются выводами h1 и h2
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

    return out_o1


network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x))  # 0.7216325609518421
```

Мы вновь получили 0.7216.

Пример тренировки нейронной сети – минимизация потерь, Часть 1

Предположим, у нас есть следующие параметры (табл. 2):

Давайте натренируем нейронную сеть таким образом, чтобы она предсказывала пол заданного человека в зависимости от его веса и роста (рис. 25).

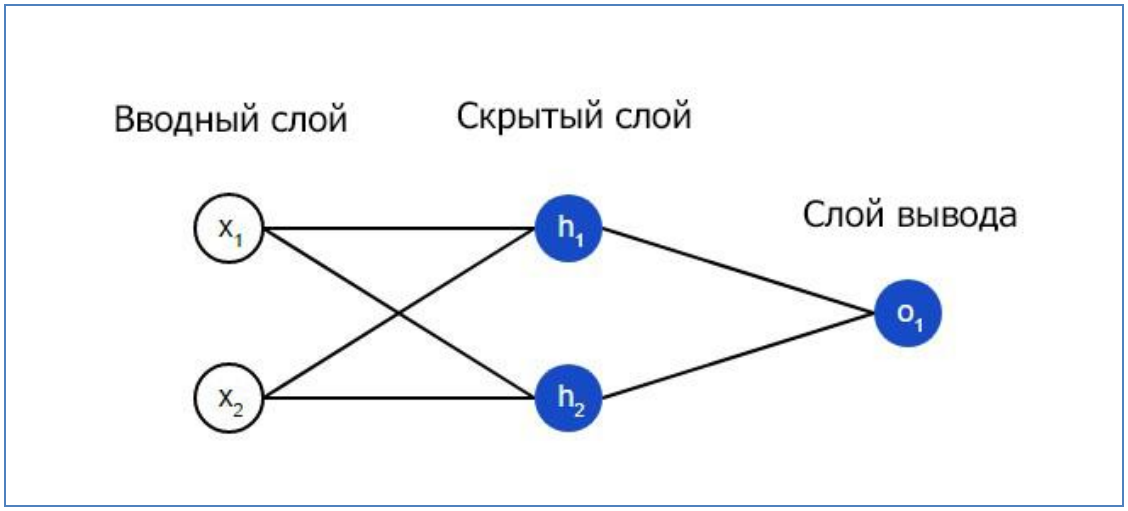

Рисунок 24 – Схема нейронной сети

Таблица 2 – Параметры

Имя/Name	Вес/Weight (фунты)	Рост/Height (дюймы)	Пол/Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Мужчины Male будут представлены как 0, а женщины Female как 1. Для простоты представления данные также будут несколько смещены (Табл. 3).

Для оптимизации здесь произведены произвольные смещения 135 и 66. Однако, обычно для смещения выбираются средние показатели.

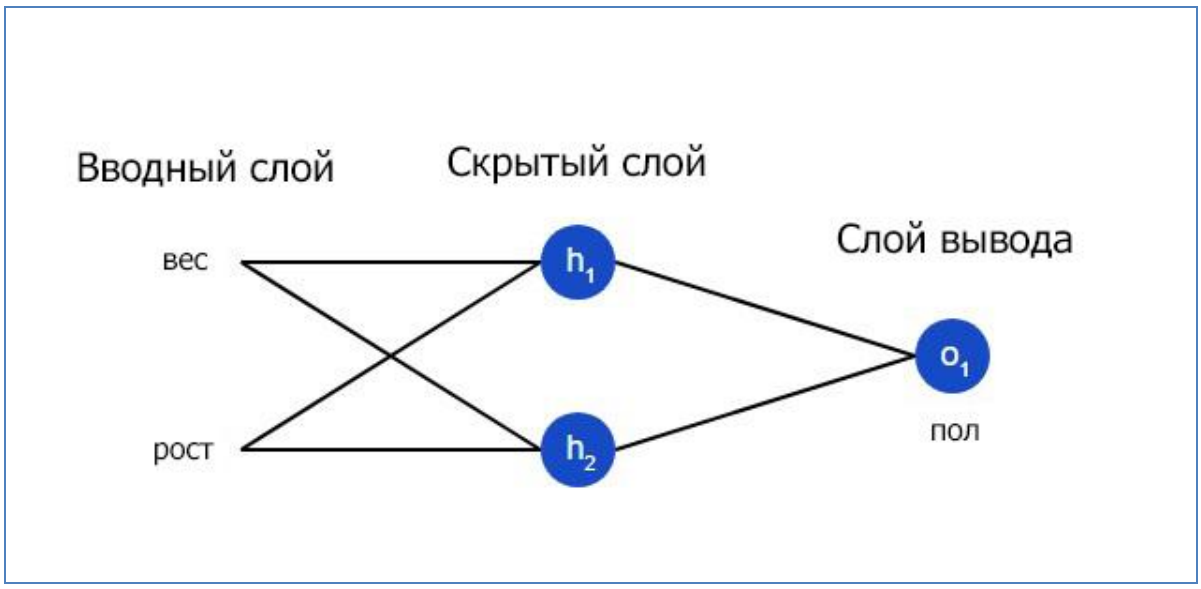

Рисунок 25 – Нейронная сеть предсказания пола заданного человека в зависимости от его веса и роста

Потери

Перед тренировкой нейронной сети потребуется выбрать способ оценки того, насколько хорошо сеть справляется с задачами. Это необходимо для ее последующих попыток выполнять поставленную задачу лучше. Таков принцип потери.

Таблица 3 – Параметры примера

Имя/Name	**Вес/Weight (минус 135)**	**Рост/Height (минус 66)**	**Пол/Gender**
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

В данном случае будет использоваться среднеквадратическая ошибка (*MSE*) потери:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Давайте разберемся:

- N – число рассматриваемых объектов, которое в данном случае равно 4. Это Alice, Bob, Charlie и Diana;
- y – переменные, которые будут предсказаны. В данном случае это пол человека;
- y_{true} – истинное значение переменной, то есть так называемый правильный ответ. Например, для Alice значение y_{true} будет 1, то есть Female;
- y_{pred} – предполагаемое значение переменной. Это результат вывода сети.

$(y_{true} - y_{pred})^2$ называют квадратичной ошибкой (MSE). Здесь функция потери просто берет среднее значение по всем квадратичным ошибкам. Отсюда и название ошибки. Чем лучше предсказания, тем ниже потери.

Лучшие предсказания – меньшие потери.

Тренировка нейронной сети – стремление к минимизации ее потерь.

Пример подсчета потерь в тренировке нейронной сети

Скажем, наша сеть всегда выдает 0 (табл. 4). Другими словами, она уверена, что все люди – Мужчины. Какой будет потеря?

Таблица 4 – Параметры подсчета потерь в тренировке нейронной сети

Имя/Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

$$MSE = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

Python – код среднеквадратической ошибки (MSE)

Ниже представлен код для подсчета потерь:

Python

```
import numpy as np


def mse_loss(y_true, y_pred):
  # y_true и y_pred являются массивами numpy с одинаковой
длиной
    return ((y_true - y_pred) ** 2).mean()


y_true = np.array([1, 0, 0, 1])
y_pred = np.array([0, 0, 0, 0])
print(mse_loss(y_true, y_pred))  # 0.5
```

При возникновении сложностей с пониманием работы кода стоит ознакомиться с guickstart в NumPy для операций с массивами.

Тренировка нейронной сети – многовариантные исчисления, Часть 2

Текущая цель понятна – это минимизация потерь нейронной сети. Теперь стало ясно, что повлиять на предсказания сети можно при помощи изменения ее веса и смещения. Однако, как минимизировать потери?

Здесь будут затронуты многовариантные исчисления.

Для простоты давайте представим, что в наборе данных рассматривается только Alice (табл. 5):

Таблица 5 – Параметры тренировки нейронной сети – многовариантные исчисления

Имя/Name	**Вес/Weight (минус 135)**	**Рост/Height (минус 66)**	**Пол/Gender**
Alice	-2	-1	1

Затем потеря среднеквадратической ошибки будет просто квадратической ошибкой для Alice:

$$MSE = \frac{1}{1}\sum_{i=1}^{1}(y_{true} - y_{pred})^2 = (y_{true} - y_{pred})^2 = (1 - y_{pred})^2$$

Еще один способ понимания потери – представление ее как функции веса и смещения. Давайте обозначим каждый вес и смещение в рассматриваемой сети (рис. 26):

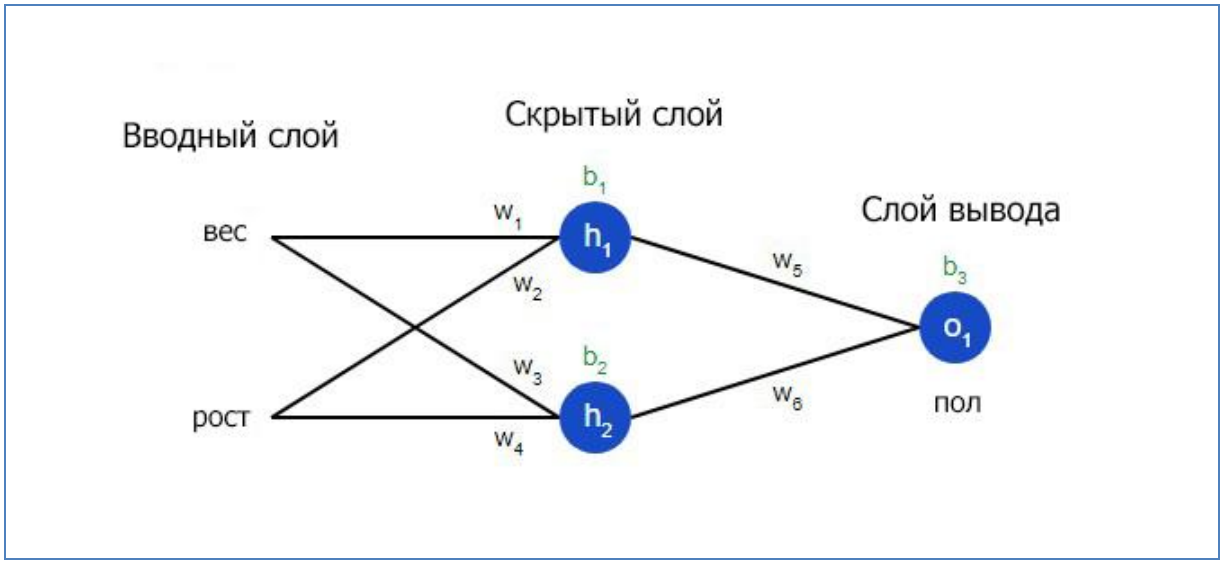

Рисунок 26 – Нейронная сеть (понимание потери – представление ее как функции веса и смещения)

Затем можно прописать потерю как многовариантную функцию:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Представим, что нам нужно немного отредактировать w_1. В таком случае, как изменится потеря L после внесения поправок в w_1?

На этот вопрос может ответить частная производная $\frac{\partial L}{\partial w_1}$.

Здесь математические вычисления будут намного сложнее.

Для начала, давайте перепишем частную производную в контексте $\frac{\partial y_{pred}}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}$$

Данные вычисления возможны благодаря дифференцированию сложной функции.

Подсчитать $\frac{\partial L}{\partial y_{pred}}$ можно благодаря вычисленной выше $L = (1 - y_{pred})^2$:

$$\frac{\partial L}{\partial y_{pred}} = \frac{\partial (1 - y_{pred})^2}{\partial y_{pred}} = \boxed{-2(1 - y_{pred}}$$

Теперь, давайте определим, что делать с $\frac{\partial y_{pred}}{\partial w_1}$. Как и ранее, позволим h_1, h_2, o_1 стать результатами вывода нейронов, которые они представляют. Дальнейшие вычисления:

$$y_{pred} = o_1 = f(w_5 h_1 + w_6 h_2 + b_3)$$

Как было указано ранее, здесь f является функцией активации сигмоида.

Так как w_1 влияет только на h_1, а не на h_2, можно записать:

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial y_{pred}}{\partial h_1} = \boxed{w_5 * f'(w_5 h_1 + w_6 h_2 + b_3)}$$

Использование дифференцирования сложной функции.

Те же самые действия проводятся для $\frac{\partial h_1}{\partial w_1}$:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b)$$

$$\frac{\partial h_1}{\partial w_1} = \boxed{x_1 * f'(w_1 x_1 + w_2 x_2 + b)}$$

Еще одно использование дифференцирования сложной функции.

В данном случае x_1– вес, а x_2 – рост. Здесь $f'(x)$ как производная функции сигмоида встречается во второй раз. Попробуем вывести ее:

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

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

Функция $f'(x)$ в таком виде будет использована несколько позже.

Теперь $\frac{\partial L}{\partial w_1}$ разбита на несколько частей, которые будут оптимальны для подсчета:

$$\boxed{\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}$$

Эта система подсчета частных производных при работе в обратном порядке известна, как метод обратного распространения ошибки, или backprop.

У нас накопилось довольно много формул, в которых легко запутаться. Для лучшего понимания принципа их работы рассмотрим следующий пример.

Пример подсчета частных производных

В данном примере (табл. 6) также будет задействована только Alice:

Таблица 6 – Параметры примера подсчета частных производных

Имя/Name	Вес/Weight (минус 135)	Рост/Height (минус 66)	Пол/Gender
Alice	-2	-1	1

Здесь вес будет представлен как 1, а смещение как 0 (табл. 5). Если выполним прямое распространение (feedforward) через сеть, получим:

$$h_1 = f(w_1 x_1 + w_2 x_2 + b_1) = f(-2 + (-1) + 0) = 0.0474$$
$$h_2 = f(w_3 x_1 + w_4 x_2 + b_2) = 0.0474$$
$$o_1 = f(w_5 h_1 + w_6 h_2 + b_3) = f(0.0474 + 0.0474 + 0) = 0.524$$

Выдачи нейронной сети $y_{pred} = 0.524$. Это дает нам слабое представление о том, рассматривается мужчина Male (0), или женщина Female (1). Давайте подсчитаем $\frac{\partial L}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial L}{\partial y_{pred}} = -2(1 - y_{pred}) = -2(1 - 0.524) = -0.952$$

$$\frac{\partial y_{pred}}{\partial h_1} = w_5 * f'(w_5 h_1 + w_6 h_2 + b_3) = 1 * f'(0.0474 + 0.0474 + 0)$$
$$= f(0.0948) * (1 - f(0.0948)) = 0.249$$

$$\frac{\partial h_1}{\partial w_1} = x_1 * f'(w_1 x_1 + w_2 x_2 + b_1) = -2 * f'(-2 + (-1) + 0) =$$
$$= -2 * f(-3) * (1 - f(-3)) = -0.0904$$

$$\frac{\partial L}{\partial w_1} = -0.952 * 0.249 * (-0.0904) = \boxed{0.0214}$$

Напоминание: мы вывели $f'(x) = f(x) * (1 - f(x))$ ранее для нашей функции активации сигмоида.

Результат говорит о том, что если мы собираемся увеличить w_1, то L немного увеличивается в результате.

7. ПРАКТИКУМ ПРИМЕНЕНИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА НА Python

Ранее мы указали, что искусственный интеллект включает в себя такие подобласти, как машинное обучение, нейронные сети, обработка естественного языка и компьютерное зрение. Рассмотрим эти области на конкретных задачах.

7.1. Обработка естественного языка и компьютерное зрение

Задача 1. Разработка чат-бота для ответов на сообщения пользователей группы ВКонтакте.

Перед началом разработки необходимо создать и настроить личную группу, а затем получить токен группы – уникальный ключ, который выдается администратору сообщества для работы с API (интерфейсом программирования приложения).

Long Polling – это технология, которая позволяет получать данные о новых событиях с помощью «длинных запросов». Сервер получает запрос, но отправляет ответ на него не сразу, а лишь тогда, когда произойдет какое-либо событие (например, придёт новое сообщение), либо истечет заданное время ожидания.

Решение:

```
# Импортируем библиотеку vk_api
import vk_api
# импортируем библиотеку longpoll
from vk_api.longpoll import VkLongPoll, VkEventType

# Создаём переменную для удобства, в которой хранится токен от
группы

token="..." # В кавычки вставляем ранее взятый из группы токен.

# Подключаем токен и longpoll
tk = vk_api.VkApi(token = token)
give = tk.get_api()
longpoll = VkLongPoll(tk)
```

```
# Создадим функцию для ответа на сообщения в сообщения группы
def fbot(id, text):
   tk.method('messages.send',  {'user_id' : id, 'message'  : text,
'random_id': 0})

# Слушаем longpoll(Сообщения)
for event in longpoll.listen():
   if event.type == VkEventType.MESSAGE_NEW:
     # Чтобы наш бот не слышал и не отвечал на самого себя
      if event.to_me:

       # Для того, чтобы бот читал все с маленьких букв
        message  = event.text.lower()
        # Получаем id пользователя
        id = event.user_id
   # Перед нами структура сообщений, на которые бот сможет
ответить, elif можно создавать сколько угодно, if и else же могут
быть только 1 в данной ситуации.
   # if - если, else – иначе (значит, бот получил сообщение, на
которое не вызвана наша функция для ответа)

       if message == 'привет':
         fbot(id, 'Привет, я бот!')

       elif message == 'как дела?':
           fbot(id, 'Хорошо, а твои как?' )

       elif message == 'поговорим о книгах':
           fbot(id, 'Какую книгу вы читали?')

       elif message == Искусственный интеллект на Питон':
         fbot(id, 'Хороший выбор!')

       elif message == 'пока':
         fbot(id, 'До связи!')
       else:
          fbot(id, 'Я вас не понимаю!')
```

Задача 2. Преобразование текстовой строки в голосовой объект с помощью встроенных библиотек Python gTTS и pyttsx3.

Первый способ основан на подключении библиотеки «Google text to speech». После распознавания можно сохранить звуковой файл.

Решение:

```
from gtts import gTTS
tts = gTTS(text="текстовая строка", lang='ru') # Создание объекта tts
tts.save("hello.mp3") # Сохранение звукового файла
```

Второй способ заключается в использовании стандартной библиотеки. Преобразованный звук можно воспроизвести при запуске приложения.

Решение:

```
import pyttsx3
engine = pyttsx3.init() # Создание объекта engine
engine.say("Привет, мир!") # Преобразование текста в речь
engine.runAndWait() # Проигрывание звука
```

Задача 3. Соединим обе задачи 1 и 2, чтобы получить голосовой чат-бот.

Для этого удобнее пользоваться библиотекой pyttsx3. Соответствующие команды распознавания добавляются в тело функции fbot().

Решение:

```
import pyttsx3
# Импортируем библиотеку vk_api
import vk_api
# Достаём из неё longpoll
from vk_api.longpoll import VkLongPoll, VkEventType

# Создаём переменную для удобства, в которой хранится токен от
группы

token="…" # В кавычки вставляем ранее взятый из группы токен.

# Подключаем токен и longpoll
```

```
tk = vk_api.VkApi(token = token)
give = tk.get_api()
longpoll = VkLongPoll(tk)

engine = pyttsx3.init() # Создание объекта engine
# Создадим функцию для ответа на сообщения в лс группы
def fbot(id, text):
   tk.method('messages.send',  {'user_id' : id, 'message'  : text,
'random_id': 0})
   engine.say(text) # Преобразование текста в речь
   engine.runAndWait() # Проигрывание звука
# Слушаем longpoll(Сообщения)
for event in longpoll.listen():
   if event.type == VkEventType.MESSAGE_NEW:
     # Чтобы наш бот не слышал и не отвечал на самого себя
      if event.to_me:

       # Для того, чтобы бот читал все с маленьких букв
        message  = event.text.lower()
        # Получаем id пользователя
        id = event.user_id

 # Основная структура сообщений

      if message == 'привет':
        out_str= 'Привет, я бот!'
        fbot(id, out_str)

      elif message == 'как дела?':
          out_str='Хорошо, а твои как?'
          fbot(id, out_str )

      elif message == 'поговорим о книгах':
          fbot(id, 'Какую книгу вы читали?')

      elif message == 'искусственный интеллект на python':
        fbot(id, 'Хороший выбор!')
```

```
elif message == 'пока':
  fbot(id, 'До связи!')
else:
   fbot(id, 'Я вас не понимаю!')
```

Задача 4. Преобразование утвердительных предложений в вопросительные с использованием модального глагола (can, will)

NLP – обработка естественного языка. Библиотека spaCy в Python содержит морфологический разбор слов английского языка. При разработке программ на выделение частей речи применяется процесс токенизации.

Токенизация – это процесс разбиения входного текста на меньшие элементы, такие как слова или предложения. Эти элементы называются токенами (лексемами).

Для работы с библиотекой spaCy необходимо не только импортировать саму библиотеку, но и загрузить основной или расширенный набор данных, например, 'en_core_web_sm'. Набор данных предварительно нужно скачать командой:

python -m spacy download en

Решение:

```
import spacy
nlp = spacy.load('en_core_web_sm')
phrase=input("Enter sentence:  ")
lastt=len(phrase)
if phrase[lastt-1]!=".":
   phrase=phrase+"."
doc = nlp(phrase)
sent = ''
for i,token in enumerate(doc):
   if token.tag_ == 'PRP' and doc[i+1].tag_ == 'MD' and doc[i+2].tag_
== 'VB':
    sent = doc[i+1].text.capitalize() + ' ' + doc[i].text
    sent = sent + ' ' + doc[i+2:].text
    break
```

```
# Повторная токенизация
doc=nlp(sent)
for i,token in enumerate(doc):
    if token.tag_ == 'PRP' and token.text == 'I':
     sent = doc[:i].text + ' you ' + doc[i+1:].text
     break

doc=nlp(sent)
for i,token in enumerate(doc):
    if token.tag_ == 'PRP$' and token.text == 'my':
     sent = doc[:i].text + ' your ' + doc[i+1:].text
     break
    if token.tag_ == 'PRP$' and token.text == 'your':
     sent = doc[:i].text + ' my ' + doc[i+1:].text
     break

doc=nlp(sent)
for i,token in enumerate(doc):
    if token.tag_ == 'VB':
     sent = doc[:i].text + ' ' + doc[i:].text
     break

doc=nlp(sent)
sent = doc[:len(doc)-1].text + '?'

print(sent)
```

Задача 5. Имитация беседы путем составления вопросительных предложений из повествовательных по шаблону: «местоимение глагол дополнение»

Решение:

```
import spacy
import sys
def find_chunk(doc):
  chunk = ''
  for i,token in enumerate(doc):
```

```
    if token.dep_ == 'dobj':
      shift = len([w for w in token.children])
      #print([w for w in token.children])
      chunk = doc[i-shift:i+1]
      break
  return chunk
def determine_question_type(chunk):
  question_type = 'yesno'
  for token in chunk:
    if token.dep_ == 'amod':
      question_type = 'info'
  return question_type

def generate_question(doc,  question_type):
  sent = ''
  for i,token in enumerate(doc):
    if token.tag_ == 'PRP' and doc[i+1].tag_ == 'VBP':
      sent = 'do ' + doc[i].text
      sent = sent + ' ' + doc[i+1:].text
      break
  doc=nlp(sent)
  for i,token in enumerate(doc):
    if token.tag_ == 'PRP' and token.text == 'I':
      sent = doc[:i].text + ' you ' + doc[i+1:].text
      break
  doc=nlp(sent)
  if question_type == 'info':
    for i,token in enumerate(doc):
      if token.dep_ == 'dobj':
        sent = 'why ' + doc[:i].text + ' one ' + doc[i+1:].text
        break
  if question_type == 'yesno':
    for i,token in enumerate(doc):
      if token.dep_ == 'dobj':
        sent = doc[:i-1].text + ' a red ' + doc[i:].text
        break
  doc=nlp(sent)
  sent = doc[0].text.capitalize() +' ' + doc[1:len(doc)-1].text + '?'
  return sent
```

```
#the main code
phrase=input("Enter sentence: ")
lastt=len(phrase)
if phrase[lastt-1]!=".":
   phrase=phrase+"."
if len(phrase) > 1:
 sent = phrase
 nlp = spacy.load('en_core_web_sm')
 doc = nlp(sent)
 chunk = find_chunk(doc)
 if str(chunk) == '':
  print('The sentence does not contain a direct object.')
  sys.exit()
 question_type = determine_question_type(chunk)
 question = generate_question(doc, question_type)
 print(question)
else:
 print('You did not submit a sentence!')
```

Задача 6. Преобразовать цветное изображение в оттенки серого с помощью библиотеки OpenCV.

Для работы с графическими изображениями необходимо загрузить библиотеку поддержки компьютерного зрения CV:

pip install opencv-python

Решение:

```
import cv2
# Считывание изображения, которое должно быть преобразовано
# в серое цветовое пространство
image1 = cv2.imread('test_img_color.png')

# Преобразование изображения в серое цветовое пространство
# с помощью функции cvtColor и сохранение полученного
изображения
imageresult = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
```

```
# Вывод результата
cv2.imshow("Start image", image1)
cv2.imshow("End image", imageresult)
cv2.waitKey(0)
```

Задача 7. Выделить контуры изображения цветом.

Решение:

```
import cv2
# загрузка изображения из файла
image = cv2.imread('robot.jpg')
# конвертация в оттенки серого
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# применение бинаризации для выделения контуров объектов
thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)[1]
# поиск и отрисовка контуров объектов
contours, hierarchy = cv2.findContours(thresh,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)
# отображение результата
 cv2.imshow('image', image)
 cv2.waitKey(0)
 cv2.destroyAllWindows()
```

Задания для самостоятельной работы.

1. Добавление в чат-бот изображения из ВК по текстовому запросу пользователя.
2. Создание чат-бота для ответов на сообщения в сети Telegram.
3. Разбиение текста книги на главы, преобразование каждой главы в речь и сохранение в виде аудиофайла с именем, содержащим номер главы.
4. Преобразование утвердительных предложений английского языка в отрицательные.
5. С помощью метода cv2.threshold преобразовать изображение в черно-белое.

7.2. Машинное обучение и нейронные сети

Задача 1. Простой перцептрон. Составить программу, в которой вручную подобрать по два варианта параметров нейрона (величину порога и синаптические силы) так, чтобы он моделировал две логические.

Функции «И» и «ИЛИ».

Нейрон МакКаллока-Питтса представляет собой формальную модель нервной клетки, состоящей из тела и нескольких отростков, называемых нервными волокнами. В модели МакКаллока-Питтса клетка – это ячейка, снабженная специальным числовым атрибутом, так называемым порогом. Из каждого нейрона выходит ребро, моделирующее нервное волокно и называемое аксоном. С помощью нейрона МакКаллока-Питтса решаются задачи моделирования логических функций «И» и «ИЛИ».

Решение:

```
# импортируем библиотеку для работы с массивами
import numpy as np

#массивы с логической функцией
X0 = np.array([0,0])
X1 = np.array([0,1])
X2 = np.array([1,0])
X3 = np.array([1,1])
# объявляем массив с весами
W1=float(input("W1="))
W2=float(input("W2="))
W = np.array([W1, W2])
# определяем пороговое значение
theta=float(input("theta="))

# функция активации
def activation_func(sum):
    if sum >= theta:
        return 1
    else:
        return 0
```

```
# вычисляем скалярное произведение векторов с помощью ф-и np.dot
# np.dot(X,W) = x0*w0 + x1*w1
y0 = np.dot(X0,W)
y1 = np.dot(X1,W)
y2 = np.dot(X2,W)
y3 = np.dot(X3,W)
# в зависимости от выходного значения ф-и активации, выводим
конъюнкцию
if activation_func(y0)==0 and activation_func(y1)==0 and activa-
tion_func(y2)==0 and activation_func(3)==1:
    print(f'конъюнкция верно  W1={W1}, W2={W2}, theta={theta}')
else:
    print('конъюнкция - нет')
# в зависимости от выходного значения ф-и активации, выводим
дизъюнкцию
if activation_func(y0)==0 and activation_func(y1)==1 and activa-
tion_func(y2)==1 and activation_func(3)==1:
    print(f'дизъюнкция верно  W1={W1}, W2={W2}, theta={theta}')
else:
    print('дизъюнкция нет')
```

Задача 2. Вывести 9 цифр обучающего набора MNIST.

Необходимо заранее загрузить файлы. Оригинальный набор данных MNIST состоит из 4 файлов:

train-images-idx3-ubyte.gz: обучающий набор изображений;
train-labels-idx1-ubyte.gz: обучающий набор меток;
t10k-images-idx3-ubyte.gz: тестовый набор изображений;
t10k-labels-idx1-ubyte.gz: тестовый набор меток.

Решение:

```
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt
```

```
pathToData = 'E:\\mnist\\' #заранее загрузить файл с dataset в
указанную папку
mndata = MNIST(pathToData) # pathToData – ранее заданный путь к
папке с данными
mndata.gz = True # Разрешаем чтение архивированных данных
imagesTrain, labelsTrain = mndata.load_training()# Обучающая
выборка (данные и метки)
imagesTest, labelsTest = mndata.load_testing()# Тестовая выборка
(данные и метки)
X_train = np.asarray(imagesTrain)
y_train = np.asarray(labelsTrain)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1) #нормализация
данных – подгонка данных к матрице для вывода в виде графика
# Выводим 9 изображений обучающего набора
names = []

for i in range(10): names.append(chr(48 + i)) # ['0', '1', '2', ..., '9']
for i in range(9):
    plt.subplot(3, 3, i + 1)
    ind = y_train[i]
    img = X_train[i][0:28, 0:28, 0]
    plt.imshow(img, cmap = plt.get_cmap('gray'))
    plt.title(names[ind])
    plt.axis('off')
plt.subplots_adjust(hspace = 0.5)
plt.show()
```

Задача 3. Распознавание рукописных цифр с помощью набора данных MNIST.

Решение состоит из двух частей. В первом файле проводится обучение модели на основе тренировочного набора данных MNIST. На выходе получим файл с обученной моделью mnist.h5. Во втором файле реализуется приложение с графическим пользовательским интерфейсом для изображения символа и его дальнейшего распознавания.

Решение:

Файл trainmodel.py:

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# скачиваем данные и разделяем на набор для обучения и тестовый
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(x_train.shape, y_train.shape)
num_classes = 10
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)

# преобразование векторных классов в бинарные матрицы
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('Размерность x_train:', x_train.shape)
print(x_train.shape[0], 'Размер train')
print(x_test.shape[0], 'Размер test')
batch_size = 128
epochs = 10

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adadelta(),metrics=['accuracy'])
hist = model.fit(x_train, y_train, batch_size = batch_size,
epochs=epochs, verbose=1, validation_data=(x_test, y_test))
print("Модель успешно обучена")

model.save('mnist1.h5')
print("Модель сохранена как mnist1.h5")
score = model.evaluate(x_test, y_test, verbose=0)
print('Потери на тесте:', score[0])
print('Точность на тесте:', score[1])
```

Файл gui_digit_recognizer.py:

```
from keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np

model = load_model('mnist.h5') #загружаем файл, полученный на
первом этапе

def predict_digit(img):
    # изменение размера изображений на 28x28
    img = img.resize((28,28))
    # конвертируем rgb в grayscale
    img = img.convert('L')
    img = np.array(img)
    # изменение размерности для поддержки модели ввода и
нормализации
    img = img.reshape(1,28,28,1)
    img = img/255.0
    # предсказание цифры
    res = model.predict([img])[0]
    return np.argmax(res), max(res)
```

```
class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        self.x = self.y = 0

        # Создание элементов
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white",
cursor="cross")
        self.label = tk.Label(self, text="Думаю..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Распознать", command =
self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Очистить", command =
self.clear_all)

        # Сетка окна
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)

        # self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
        self.canvas.delete("all")

    def classify_handwriting(self):
        HWND = self.canvas.winfo_id()
        rect = win32gui.GetWindowRect(HWND)  # получаем
координату холста
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')

    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
```

```
        r=8
        self.canvas.create_oval(self.x-r,  self.y-r, self.x + r, self.y + r,
fill='black')

app = App()
mainloop()
```

Задания для самостоятельной работы.

1. Создать нейрон для реализации логической функции «исключающее или».

2. С помощью набора данных для распознавания цифр MNIST вывести первую и последнюю цифры набора, а также все рукописные варианты цифры «5».

3. С помощью набора данных для распознавания предметов fashion распознать и вывести ботинки.

СПИСОК ЛИТЕРАТУРЫ

1. Антонио Д. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow / Д. Антонио, П. Суджит ; перевод с английского А.А. Слинкин. – Москва : ДМК Пресс, 2018. – 294 с.
2. Асадуллаев Р.Г. Нечеткая логика и нейронные сети : учебное пособие ;/ сост. Р.Г. Асадуллаев. – Белгород, 2017. – 309 с.
3. Барский А.Б. Логические нейронные сети : учебное пособие / А.Б. Барский. – Москва : Бином, 2013. – 352 с.
4. Бессмертный И.А. Искусственный интеллект / И.А. Бессмертный – Санкт-Петербург : СПбГУ ИТМО, 2010. – 132 с.
5. Вандер П. Python для сложных задач: наука о данных и машинное обучение / П. Вандер. – Санкт-Петербург : Питер, 2018. – 576 с.
6. Васильев Ю. Обработка естественного языка. Python и spaCy на практике / Ю. Васильев. – Санкт-Петербург : Питер, 2021. – 256 с.
7. Вирсански Э. Генетические алгоритмы на Python ; пер. с англ. А.А. Слинкина. – Москва : ДМК Пресс, 2020. – 286 с.
8. Воронов М.В. Системы искусственного интеллекта : учебник и практикум для вузов / М.В. Воронов, В.И. Пименов, И.А. Небаев. – Москва : Издательство Юрайт, 2022. – 256 с.
9. Галушкин А.И. Нейронные сети: история развития теории : учебное пособие для вузов. / А.И. Галушкин, Я.З. Цыпкин. – Москва : Альянс, 2015. – 840 с.
10. Галушкин А.И. Нейронные сети: основы теории. / А.И. Галушкин. – Москва : РиС, 2015. – 496 с.
11. Гудфеллоу Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль – пер. с анг. А. А. Слинкина. – 2-е изд., испр. – Москва : ДМК Пресс, 2018. – 652 с.
12. Дейтел П. Python: Искусственный интеллект, большие данные и облачные вычисления / П. Дейтел, Х. Дейтел. – Санкт-Петербург : Питер, 2020. – 864 с.
13. Джоши П. Искусственный интеллект с примерами на Python ; пер. с англ. – Санкт-Петербург: ООО «Диалектика», 2019. – 448 с.
14. Душкин Р.В. Искусственный интеллект / Р.В. Душкин. – Москва : ДМК Пресс, 2019. – 280 с.

15. Искусственный интеллект на Python с использованием TensorFlow и Keras [Электронный ресурс]. – Режим доступа URL: https://habr.com/ru/articles/770554/ (дата обращения 05.01.2024 г.).

16. Каллан Р. Нейронные сети: Краткий справочник / Р. Каллан. – Москва : ООО «И.Д. Вильямс», 2017. – 288 с.

17. Комашинский В.И. Нейронные сети и их применение в системах управления и связи / В.И. Комашинский, Д.А. Смирнов. – Москва : ГЛТ, 2003. – 94 с.

18. Коэльо Л. Построение систем машинного обучения на языке Python. / Л. Коэльо, В. Ричарт. – 2-е изд. ; пер. с англ. А.А. Слинкин. – Москва : ДМК Пресс, 2016. – 302 с.

19. Люгер Дж. Искусственный интеллект: стратегии и методы решения сложных проблем. – 4-е изд. ; пер. с англ. – Москва : Издательский дом «Вильяме», 2003. – 864 с.

20. Макартичян С.В. Нейронные системы обработки информации : учебно-методическое пособие / С.В. Макартичян, Н.С. Кузнецова, С.С. Дементьев ; ВолгГТУ. – Волгоград, 2020. – 108 с.

21. Николенко С. Глубокое обучение / С. Николенко, А. Кадурин, Е. Архангельская. – Санкт-Петербург : Питер, 2018. – 480 с.: ил.

22. Новиков Ф.А. Искусственный интеллект: представление знаний и методы поиска решений: учебное пособие / Ф.А. Новиков. – Санкт-Петербург : Изд-во Политехн. ун-та, 2010. – 240 с.

23. Паттерсон Дж. Глубокое обучение с точки зрения практика / Дж. Паттерсон, А. Гибсон ; пер. с анг. А.А. Слинкина. – Москва : ДМК Пресс, 2018. – 418 с.

24. Постолит А.В. Основы искусственного интеллекта в примерах на Python. Самоучитель / А.В. Постолит. – Санкт-Петербург : БХВ-Петербург, 2021. – 448 с.

25. Протодьяконов А.В. Алгоритмы Data Science и их практическая реализация на Python : учебное пособие / А.В. Протодьяконов, П.А. Пылов, В.Е. Садовников. – Москва ; Вологда : Инфра-Инженерия, 2022. – 392 с.

26. Рашид Т. Создаем нейронную сеть ; пер. с англ. / Т. Рашид. – Санкт-Петербург : ООО «Альфа-книга», 2017. – 272 с.

27. Редько В.Г. Эволюция, нейронные сети, интеллект: Модели и концепции эволюционной кибернетики / В.Г. Редько. – Москва : Ленанд, 2019. – 224 с.

28. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Москва : РиС, 2013. – 384 с.

29. Усков А.А. Интеллектуальные технологии управления. Искусственные нейронные сети и нечеткая логика. / А.А. Усков, А.В. Кузьмин. – Москва : Горячая линия – Телеком, 2004. – 143 с.

30. Фостер Д. Генеративное глубокое обучение. Творческий потенциал нейронных сетей / Д. Фостер. – Санкт-Петербург : Питер, 2020. – 336 с.

31. Хайкин С. Нейронные сети: полный курс / С. Хайкин. – Москва : Диалектика, 2019. – 1104 с.

32. Шалев-Шварц Ш. Идеи машинного обучения: от теории к алгоритмам / Ш. Шалев-Шварц, Ш. Бен-Давид ; пер. с англ. А.А. Слинкина. – Москва : ДМК Пресс, 2019. – 436 с.

33. Ширяев В.И. Финансовые рынки: Нейронные сети, хаос и нелинейная динамика / В.И. Ширяев. – Москва : КД Либроком, 2016. – 232 с.

34. Я. Омельяненко Я. Эволюционные нейросети на языке Python ; пер. с анг. В. С. Яценкова. – Москва : ДМК Пресс, 2020. – 310 с.

35. Ясницкий Л.М. Искусственный интеллект. Элективный курс: методическое пособие / Л.М. Ясницкий, Ф.М. Черепанов. – Москва : БИНОМ. Лаборатория знаний, 2012. – 216 с.

36. Яхъяева, Г.Э. Нечеткие множества и нейронные сети : учебное пособие / Г.Э. Яхъяева. – Москва : БИНОМ. ЛЗ, ИНТУИТ.РУ, 2012. – 316 с.

37. 10 библиотек Python для машинного обучения и искусственного интеллекта [Электронный ресурс]. – Режим доступа URL: https://uproger.com/10-bibliotek-python-dlya-mashinnogo-obucheniya/?ref=vc.ru (дата обращения 05.01.2024 г.).

СОДЕРЖАНИЕ

Учебное издание

Андропова Ольга Юрьевна,
Васильева Ирина Ивановна,
Гнездилова Наталия Александровна

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЯЗЫК ПРОГРАММИРОВАНИЯ Python

Учебное пособие

Технический редактор – Г. Н. Бурганская
Техническое исполнение – В. М. Гришин
Книга публикуется в авторской редакции

Лицензия на издательскую деятельность
ИД № 06146. Дата выдачи 26.10.01.
Формат 60 x 84 /16. Гарнитура Times. Печать трафаретная.
Печ.л. 6,7 Уч.-изд.л. 6,5
Тираж 300 экз. Заказ 8

Отпечатано с готового оригинал-макета на участке оперативной полиграфии
Елецкого государственного университета им. И. А. Бунина
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Елецкий государственный университет им. И.А. Бунина»
399770, г. Елец, ул. Коммунаров, 28,1