0.1 Введение в искусственный интеллект и нейронные сети . . 4
0.1.1 Основные задачи ИИ 4
0.1.2 Исторический обзор развития ИНС 4
0.1.3 Биологическая модель нейрона 5
0.1.4 Ключевые понятия 6
0.2 Основы нейронных сетей 7
0.2.1 Области искусственного интеллекта и модели внешней среды 7
0.2.2 Методы машинного и глубокого обучения 9
0.2.3 Концептуальные различия между методами машинного и глубокого обучения 10
0.2.4 Формальная постановка задач обучения 12
0.2.5 Кодирование категориальных данных 12
0.2.6 Кодирование изображений 13
0.2.7 Классификация искусственных нейронов и сетей . . 13
0.3 Подготовка данных и предобработка 15
0.3.1 Кодирование числовых и логических признаков . . 15
0.3.2 Кодирование категориальных данных 15
0.3.3 Кодирование изображений 16
0.4 Типы обучения 16

0.4.1 Обучение с учителем 16
0.4.2 Обучение без учителя 26
0.4.3 Динамическое программирование 32
0.5 Методы обучения нейронных сетей 36
0.5.1 Функции потерь и их назначение 36
0.5.2 Алгоритм градиентного спуска 37
0.5.3 Обратное распространение ошибки (Backpropagation) 38
0.5.4 Обратное распространение ошибки во времени (BPTT) для RNN 40
0.5.5 Методы нормализации входных данных и регуляризации параметров модели 42
0.5.6 Рекуррентные нейронные сети и LSTM 44
0.6 Современные архитектуры нейронных сетей 46
0.6.1 Сверточные нейронные сети (CNN) 46
0.6.2 Рекуррентные нейронные сети (RNN), LSTM и GRU 47
0.6.3 Генеративно-состязательные сети (GAN) 48
0.6.4 Трансформеры 50
0.7 Практические примеры применения ИНС и современные инструменты 52
0.7.1 Примеры практических задач 52
0.7.2 Современные инструменты и фреймворки 55
0.7.3 Применение ИНС агентом 56
0.8 Ограничения нейросетевого подхода и перспективы развития 57
0.8.1 Требования к данным и вычислительным ресурсам 57
0.8.2 Интерпретируемость моделей 57
0.8.3 Переобучение и обобщающая способность 57

0.8.4 Проблемы градиентного спуска 57
0.8.5 Перспективы развития 58

0.1 Введение в искусственный интеллект и нейронные сети

Искусственный интеллект (ИИ) — междисциплинарная область, объединяющая теорию, методы и практические системы, способные имитировать когнитивные функции человека, такие как восприятие, рассуждение, обучение и принятие решений.

0.1.1 Основные задачи ИИ

В рамках ИИ решаются задачи, связанные с:

- решением плохо формализуемых задач, где отсутствуют точные алгоритмы;
- моделированием поведения интеллектуальных агентов, способных адаптироваться к изменяющимся условиям;
- представлением и обработкой знаний для проведения логического и вероятностного анализа;
- взаимодействием с внешней средой, где система способна воспринимать сигналы и реагировать на них.

0.1.2 Исторический обзор развития ИНС

Развитие искусственных нейронных сетей можно условно разделить на несколько этапов:

1. **Начало исследований (1950-е годы).**
    - Модель персептрона Розенблатта представляла собой простейший алгоритм бинарной классификации.

- Ограничения однослойных моделей (невозможность разделения нелинейно разделимых данных) стали стимулом для дальнейших исследований.

2. **Появление многослойных сетей и алгоритма обратного распространения (1980-е годы).**
   - Разработка алгоритма обратного распространения ошибки позволила обучать сети с несколькими скрытыми слоями.
   - Появление теории об универсальной аппроксимации, доказывающей, что многослойная сеть может аппроксимировать любую непрерывную функцию.
3. **Эра глубокого обучения (2010-е годы и далее).**
   - Использование больших наборов данных, усиление вычислительных мощностей и развитие новых архитектур (сверточные, рекуррентные, трансформеры) привели к бурному росту эффективности ИНС.
   - Современные методы позволяют решать сложнейшие задачи в распознавании образов, обработке естественного языка и других областях.

0.1.3 Биологическая модель нейрона

Биологический нейрон состоит из трёх основных частей:

- **Дендриты.** Разветвлённые отростки, собирающие сигналы от других нейронов.
- **Сома (тело нейрона).** Центральная часть, где происходит интеграция входных сигналов и принятие решения о генерации импульса.
- **Аксон.** Длинный отросток, по которому передаётся электрический импульс к другим нейронам через синапсы.

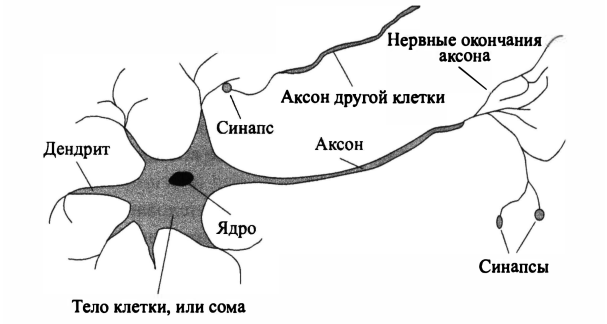

Рис. 1: Части нервной клетки или нейрона. Каждый нейрон состоит из тела клетки (или сомы), которое содержит ядро клетки.

Синаптическая передача сигналов и пластичность (способность изменять силу связей) являются ключевыми для процессов обучения в мозге. Под синаптической передачей в ИНС метафорически понимается передача сигнала от одного нейрона к другому. Сила связей — это веса между нейронами, которые определяют вклад одного нейрона в активацию другого. Эти веса обучаются.

0.1.4 Ключевые понятия

- **Плохоформализуемая задача.** Задача, для которой невозможно составить полное и точное формальное описание из-за высокой сложности входных данных или неопределенности условий. Такие задачи требуют использования эвристических и адаптивных методов [1].

- **Агент.** Автономная система, способная воспринимать информацию из внешней среды, анализировать её и принимать решения. Примеры: программные роботы, автономные транспортные средства [2].

- **Внешняя среда.** Совокупность всех факторов, с которыми взаимодействует агент, включая физические объекты и информационные потоки.

- **Поведение.** Совокупность действий, предпринимаемых агентом в ответ на внешние сигналы, определяемых внутренними алгоритма-

ми.

- **Модель.** Формальное представление системы, позволяющее проводить симуляцию, прогнозирование и анализ. Модели могут быть логическими, вероятностными, дифференциальными или нейронными [3].

- **Обучение.** Процесс настройки параметров модели на основе данных для уменьшения ошибки предсказания. Может осуществляться с учителем, без учителя или с подкреплением.

- **Знание.** Систематизированная информация для принятия решений, представляемая декларативно или процедурно.

- **Искусственный нейрон.** Элемент нейронной сети, имитирующий биологический нейрон, суммирующий входные сигналы с весовыми коэффициентами и применяющий нелинейную функцию активации [4].

- **Искусственная нейронная сеть (ИНС).** Система взаимосвязанных нейронов, способная моделировать сложные зависимости и решать задачи классификации, регрессии, аппроксимации и оптимизации [3].

0.2 Основы нейронных сетей

0.2.1 Области искусственного интеллекта и модели внешней среды

Искусственный интеллект находит широкое применение в различных областях. Каждая из этих областей требует специфического подхода к моделированию окружающей среды и использованию различных методов машинного обучения.

- **Компьютерное зрение**:

- **Модели среды**: изображения, видео, 3D-сцены. Эти данные представляют собой визуальные объекты, которые нужно анализировать с помощью ИИ.
- **Примеры задач**: классификация изображений (например, классификация объектов на фотографиях), обнаружение объектов (выделение конкретных объектов на изображении), сегментация (разделение изображения на смысловые части).
- **Методы**: Свёрточные нейронные сети (CNN), такие как `ResNet`, `EfficientNet`, а также трансформеры для обработки изображений (`ViT`).

- **Обработка естественного языка (NLP)**:

  Модели среды: текстовые последовательности, языковые структуры. В этом случае данные — это текстовые строки или целые документы.**Примеры задач**: машинный перевод (перевод текста с одного языка на другой), анализ тональности (определение эмоциональной окраски текста), генерация текста (создание текста по заданной теме). **Методы**: Рекуррентные нейронные сети (RNN), LSTM для обработки последовательностей, а также трансформеры (`BERT`, `GPT`).

- **Робототехника и управление**:
  - **Модели среды**: физические системы, сенсорные данные. Моделируются данные о положении и движении роботов, а также их взаимодействие с окружающей средой.
  - **Примеры задач**: навигация (перемещение робота по пространству), манипуляция объектами (например, захват и перемещение объектов).
  - **Методы**: обучение с подкреплением (например, Q-обучение), имитационное обучение (для тренировки роботов в симуляторах).

- **Рекомендательные системы**:

- **Модели среды**: пользовательские предпочтения, история взаимодействий. Примером являются данные о том, что пользователь смотрел или покупал в прошлом.
- **Примеры задач**: прогнозирование рейтингов (например, что пользователь может оценить высоко), персонализация контента (выдача рекомендаций на основе интересов).
- **Методы**: матричная факторизация, глубокие рекомендательные системы.

0.2.2 Методы машинного и глубокого обучения

Для решения задач, связанных с моделированием внешней среды, используются различные подходы машинного обучения. Эти методы можно разделить на несколько категорий, каждая из которых решает определённые типы задач.

- **Классическое машинное обучение**:
  - Линейные модели: используются для задач, где данные поддаются линейной аппроксимации. Пример — линейная регрессия или SVM для классификации.
  - Ансамблевые методы: такие как Random Forest и XGBoost, которые комбинируют несколько моделей для улучшения общей производительности.
  - Пример задачи: прогнозирование цен на недвижимость на основе различных факторов.
- **Глубокое обучение**:
  - Нейронные сети, такие как MLP, CNN и RNN, которые могут анализировать сложные данные (например, изображения или текст).

  - Пример задачи: распознавание речи или изображений, где требуется высокая точность.

- **Обучение с подкреплением**:
  - Модели обучения с подкреплением, такие как Q-обучение и методы политики градиента, обучают агента через взаимодействие с окружающей средой.
  - Пример задачи: обучение робота ходьбе или игры в игры, где агент должен получать награды или наказания за свои действия.

0.2.3 Концептуальные различия между методами машинного и глубокого обучения

Методы машинного и глубокого обучения, хотя и тесно связаны, имеют фундаментальные концептуальные различия в подходах к обучению и обработке данных:

- **Машинное обучение** традиционно включает модели, которые требуют предварительной обработки данных и явного извлечения признаков. Эти методы ориентированы на обработку структурированных данных, таких как табличные данные. Важным этапом в процессе является выбор и трансформация признаков, что требует экспертных знаний в предметной области.
- **Глубокое обучение**, в отличие от машинного, использует многослойные нейронные сети, которые автоматически извлекают признаки из необработанных данных (например, изображения или текст). Это позволяет применять глубокие модели к более сложным и высокоразмерным данным, таким как изображения, видео или звуковые файлы, без необходимости вручную выделять признаки.

- **Классическое машинное обучение** более эффективно в задачах с ограниченными данными и меньшими вычислительными затратами. Однако глубокие модели, несмотря на свою большую сложность и потребность в больших объёмах данных, обладают выдающимися возможностями в области обработки больших и сложных наборов данных. Например, для задачи классификации изображений свёрточные нейронные сети (CNN) могут дать результаты, значительно превосходящие традиционные методы.

- **Гибкость и универсальность**: методы глубокого обучения универсальны и могут применяться в широком спектре задач, включая задачи, в которых трудно выделить явные признаки. Например, трансформеры, использующиеся в NLP, продемонстрировали возможности обработки различных типов данных, от текстов до изображений, с минимальной настройкой.

- **Обучение с подкреплением**, в отличие от других методов, ориентировано на задачу обучения агентов через взаимодействие с окружающей средой. Это отличается от традиционных методов обучения, где модель обучается на заранее размеченных данных. Обучение с подкреплением предполагает, что агент должен принимать решения на основе текущего состояния и получать вознаграждение за свои действия.

- **Интерпретируемость**: традиционные методы машинного обучения часто имеют лучшие возможности для интерпретации и понимания работы модели, поскольку каждый этап построения модели и её принятия решения может быть прослежен и объяснён. В случае с глубоким обучением модели могут быть трудно интерпретируемыми, особенно если они включают тысячи слоёв и параметров.

0.2.4 Формальная постановка задач обучения

Для корректного обучения нейронной сети важна правильная обработка входных данных:

- Если признаки логические, то каждому признаку соответствует один узел, принимающий значение 0 или 1. Иногда используют значения -1 и $+1$ для центровки данных.
- Числовые признаки могут использоваться в исходном виде или масштабироваться с помощью нормализации (например, приведение к диапазону [0,1] или стандартизация с вычитанием среднего и делением на стандартное отклонение). Стандартизация — это способ предварительной обработки данных, при котором каждое значение пикселя преобразуется так, чтобы вся выборка имела:
  - среднее значение (mean) ≈ 0;
  - стандартное отклонение (std) ≈ 1.

  Формула стандартизации:

  $$x' = \frac{x - \mu}{\sigma}$$

  где x — исходное значение пикселя, μ — среднее по датасету, σ — стандартное отклонение.
  Это ускоряет и стабилизирует обучение, делая градиентный спуск более эффективным.
- При сильном разбросе значений может применяться логарифмическое преобразование.

0.2.5 Кодирование категориальных данных

Для категориальных признаков, имеющих более двух возможных значений, часто применяется метод one-hot кодирования. Пусть признак

принимает одно из d значений, тогда он представляется в виде вектора:

$$\mathbf{x} = [x_1, x_2, \ldots, x_d],$$

где $x_i = 1$ для истинного значения и $x_j = 0$ для всех остальных $j \neq i$.

0.2.6 Кодирование изображений

Изображения обычно представляются как матрицы (или тензоры) числовых значений, где каждый элемент соответствует интенсивности пикселя. Важно учитывать не только отдельные пиксели, но и их пространственную структуру. Для этого используются сверточные нейронные сети, где применяется операция свертки:

$$z_{i,j} = \sum_{u,v} k_{u,v}\, x_{i+u,\, j+v},$$

где $k_{u,v}$ — элементы ядра свертки. Эта операция позволяет обнаруживать локальные признаки, такие как границы, текстуры и другие особенности изображения. В свёрточной нейронной сети ядро свёртки (матрица весов) скользит по изображению, извлекая локальные признаки.

0.2.7 Классификация искусственных нейронов и сетей

Искусственные нейроны можно классифицировать по различным характеристикам:

- **По функции активации.**
  - Линейные: используются в простых моделях, где входы комбинируются без нелинейного преобразования.
  - Сигмоидальные: имеют S-образную кривую, часто используются для бинарной классификации.

- Гиперболический тангенс (tanh): похож на сигмоиду, но симметричен относительно нуля.
- ReLU (Rectified Linear Unit): обеспечивает быстрые вычисления и помогает избегать проблемы исчезающих градиентов.
- Softplus: гладкая версия ReLU, которая также находит применение в некоторых моделях.

- **По архитектуре сети.**
  - Однослойные сети (персептрон).
  - Многослойные персептроны (MLP) с одним или несколькими скрытыми слоями.
  - Сверточные нейронные сети (CNN), используемые в обработке изображений.
  - Рекуррентные нейронные сети (RNN) для обработки последовательностей. В контексте рекуррентных нейросетей (RNN) и трансформеров — это временные или логические последовательности данных: тексты, аудио, временные ряды и т.д.
  - Гибридные модели, объединяющие различные архитектурные подходы.

  MLP (многослойный перцептрон) — полностью связная архитектура, где каждый нейрон связан со всеми нейронами следующего слоя. CNN (свёрточная нейросеть) использует локальные связи через свёртки и веса ядра, что делает её эффективной для обработки изображений и сохраняет пространственную структуру данных.

- **По способу обучения.**
  - Обучение с учителем, когда используется размеченная выборка.
  - Обучение без учителя, где сеть ищет скрытые закономерности в неразмеченных данных.

– Обучение с подкреплением, когда агент получает обратную связь в виде вознаграждения. В рамках обучения с подкреплением агент может быть реализован с использованием ИНС. Нейросеть в этом случае служит для принятия решений или оценки ценности действий в различных состояниях среды.

0.3 Подготовка данных и предобработка

0.3.1 Кодирование числовых и логических признаков

Для корректного обучения нейронной сети важна правильная обработка входных данных:

- Если признаки логические, то каждому признаку соответствует один узел, принимающий значение 0 или 1. Иногда используют значения -1 и $+1$ для центровки данных.
- Числовые признаки могут использоваться в исходном виде или масштабироваться с помощью нормализации (например, приведение к диапазону [0,1] или стандартизация с вычитанием среднего и делением на стандартное отклонение).
- При сильном разбросе значений может применяться логарифмическое преобразование.

0.3.2 Кодирование категориальных данных

Для категориальных признаков, имеющих более двух возможных значений, часто применяется метод one-hot кодирования. Пусть признак принимает одно из d значений, тогда он представляется в виде вектора:

$$\mathbf{x} = [x_1, x_2, \ldots, x_d],$$

где $x_i = 1$ для истинного значения и $x_j = 0$ для всех остальных $j \neq i$.

0.3.3 Кодирование изображений

Изображения обычно представляются как матрицы (или тензоры) числовых значений, где каждый элемент соответствует интенсивности пикселя. Важно учитывать не только отдельные пиксели, но и их пространственную структуру. Для этого используются сверточные нейронные сети, где применяется операция свертки:

$$z_{i,j} = \sum_{u,v} k_{u,v}\, x_{i+u,\, j+v},$$

где $k_{u,v}$ — элементы ядра свертки. Эта операция позволяет обнаруживать локальные признаки, такие как границы, текстуры и другие особенности изображения.

0.4 Типы обучения

0.4.1 Обучение с учителем

При обучении с учителем модель получает размеченные данные $\{(x_i, y_i)\}_{i=1}^{N}$. Алгоритм обратного распространения ошибки включает следующие этапы:

1. **Прямой проход:** Каждый входной вектор x проходит через все слои сети, и вычисляется выход $\hat{y} = f(x; \theta)$. На каждом слое происходит вычисление линейной комбинации входов с весами, за которой следует применение функции активации (например, ReLU, сигмоида).

2. **Вычисление ошибки:** Вычисляется функция потерь $L(\hat{y}, y)$, например, квадратичная ошибка для задач регрессии или перекрестная энтропия для задач классификации. Ошибка отражает разницу между предсказанным значением и истинным ответом.

3. **Обратный проход:** С использованием цепного правила вычисляются производные функции потерь по каждому весу w в сети.

Пусть z — взвешенная сумма входов, тогда для одного нейрона:

$$\frac{\partial L}{\partial w_i} = -(y - \hat{y}) \cdot g'(z) \cdot x_i.$$

Здесь $g'(z)$ — производная функции активации, а x_i — входной сигнал.

4. **Обновление весов:** После вычисления градиентов веса обновляются согласно правилу:

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{\partial L}{\partial w_i},$$

где η — скорость обучения. Таким образом, веса корректируются в направлении уменьшения ошибки.

Данный процесс повторяется для всех обучающих примеров, что приводит к постепенному уменьшению ошибки сети.

Пример задачи классификации: Распознавание рукописных цифр (набор MNIST). В данном случае $\mathbf{x}$ — изображение, представляемое в виде вектора длины 784, а $y \in \{0, 1, \ldots, 9\}$. Цель — построить модель, которая, получив изображение, правильно определяет цифру.

Методы обучения с учителем

Задача регрессии — это задача машинного обучения, в которой требуется предсказать численное (не категориальное) значение на основе входных данных. Она возникает при попытке смоделировать непрерывную зависимость между переменными, например, предсказание температуры, стоимости жилья и т.п.

Линейная регрессия — один из самых ранних и простых методов статистики, моделирующий линейную зависимость между признаками и целевой переменной. В ИНС она используется как базовый пример регрессионной задачи.

1. Линейная регрессия (Regression) Модель линейной регрессии имеет вид:

$$\hat{y} = w_0 + \sum_{j=1}^{n} w_j x_j = \mathbf{w}^\top \mathbf{x} + w_0,$$

где $\mathbf{w} \in \mathbb{R}^n$ — вектор весов. Функция потерь (среднеквадратичная ошибка) определяется как:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right)^2.$$

Обучение производится методом градиентного спуска, где обновление весов происходит по правилу:

$$w_j \leftarrow w_j - \alpha \frac{\partial L}{\partial w_j}, \quad j = 0, 1, \ldots, n.$$

Пример вычисления производной:

$$\frac{\partial L}{\partial w_j} = -\frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}.$$

2. Логистическая регрессия (Logistic Regression) Для бинарной классификации используется логистическая регрессия. Модель задаётся как:

$$z = w_0 + \sum_{j=1}^{n} w_j x_j, \quad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}},$$

где $\sigma(z)$ — сигмоидальная функция. Функция потерь (бинарная кросс-энтропия):

$$L(y, \hat{y}) = -\Big[y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \Big].$$

Градиенты вычисляются по аналогии с линейной регрессией, но с учётом нелинейности $\sigma(z)$. Для параметра w_j имеем:

$$\frac{\partial L}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}.$$

3. Деревья решений и ансамблевые методы Деревья решений строят модель в виде иерархической структуры, где каждый узел выполняет разбиение пространства признаков. Алгоритм CART (Classification and Regression Trees) минимизирует ошибку разбиения, например, с использованием критерия Джини или энтропии.

$$Gini = 1 - \sum_{k=1}^{K} p_k^2, \quad H = -\sum_{k=1}^{K} p_k \log p_k,$$

где p_k — доля объектов в узле, принадлежащих классу k. Ансамблевые методы, такие как случайный лес (Random Forest) и градиентный бустинг (Gradient Boosting), комбинируют несколько деревьев для повышения точности и устойчивости модели.

4. Нейронные сети и обратное распространение ошибки Многослойный персептрон (MLP) представляет собой композицию линейных преобразований и нелинейных функций активации:

$$\mathbf{h}^{(l)} = g^{(l)}\Big(W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\Big), \quad l = 1, 2, \ldots, L,$$

где $\mathbf{h}^{(0)} = \mathbf{x}$ — вход, $\mathbf{h}^{(L)} = \hat{\mathbf{y}}$ — выход. Функция потерь L (например, перекрёстная энтропия для классификации) минимизируется методом градиентного спуска с использованием алгоритма обратного распространения ошибки (backpropagation).

Обратное распространение основано на правиле цепочки. Если рассматривать скрытый слой, то градиент ошибки по его входу:

$$\frac{\partial L}{\partial \mathbf{z}^{(l)}} = \Big(W^{(l+1)}\Big)^{\top} \frac{\partial L}{\partial \mathbf{h}^{(l+1)}} \odot g'^{(l)}\Big(\mathbf{z}^{(l)}\Big),$$

где $\mathbf{z}^{(l)} = W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ и $\odot$ — поэлементное умножение. Пример: если

используется функция ReLU, то

$$g'(z) = \begin{cases} 1, & \text{если } z > 0, \\ 0, & \text{иначе.} \end{cases}$$

Примеры применения обучения с учителем

- **Распознавание образов.** Сверточные нейронные сети (CNN) используются для классификации изображений, таких как определение объектов на фотографиях. Архитектуры типа AlexNet, VGG, ResNet показывают высокую точность на задачах компьютерного зрения.
- **Обработка естественного языка.** Рекуррентные нейронные сети (RNN) и трансформеры применяются для машинного перевода, анализа тональности текста и других задач NLP.
- **Прогнозирование временных рядов.** Линейная регрессия или LSTM используются для предсказания значений временных рядов, например, цен на акции или погодных условий.

Распознавание образов

Общее описание задачи Распознавание образов включает в себя автоматическое определение и классификацию объектов на изображениях. Ключевая цель — построение модели, способной принимать на вход изображение и выдавать корректную метку класса или локализацию объектов.

Методы и архитектуры Для распознавания образов широко применяются сверточные нейронные сети (CNN), поскольку они эффективно извлекают пространственные признаки. Основные этапы решения задачи:

1. **Предварительная обработка:** входное изображение нормализуется и преобразуется в тензор, часто с приведением значений к диапазону $[0, 1]$ или стандартизацией.

2. **Извлечение признаков:** сверточные слои применяют фильтры (ядра свертки) для выявления локальных признаков, таких как края, текстуры, углы. Пусть входное изображение представлено в виде матрицы $X \in \mathbb{R}^{H \times W \times C}$, где H и W — высота и ширина, а C — число каналов. Операция свертки для одного фильтра определяется как:

$$(X * K)(i, j) = \sum_{u=1}^{k_H} \sum_{v=1}^{k_W} \sum_{c=1}^{C} K(u, v, c)\, X(i + u - 1, j + v - 1, c),$$

   где K — ядро свертки размером $k_H \times k_W \times C$.
3. **Нелинейные преобразования:** после каждой свертки применяется функция активации (например, ReLU), что позволяет моделировать нелинейные зависимости.
4. **Субдискретизация:** с помощью операций пуллинга (например, max-pooling) уменьшается размерность признакового пространства, что помогает снизить вычислительную сложность.
5. **Классификация:** извлечённые признаки передаются в один или несколько полносвязных слоёв, завершающихся softmax-слоем для получения вероятностного распределения по классам.

Математическая формализация Пусть на входе подаётся изображение, преобразованное в вектор признаков $\mathbf{x}$, а модель представляет собой композицию функций:

$$h(\mathbf{x}) = f_{\text{fc}} \circ f_{\text{pool}} \circ f_{\text{conv}}(\mathbf{x}),$$

где:

- f_{conv} — операция свертки с несколькими фильтрами,
- f_{pool} — операция пуллинга (субдискретизации),
- f_{fc} — полносвязные слои с функцией активации и softmax на выходе.

Функция softmax, используемая на выходном слое, задаётся как:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}},$$

где z_i — выходная оценка для i-го класса, а d — общее число классов.

Обработка естественного языка (Natural Language Processing, NLP)

Общее описание задачи Обработка естественного языка включает анализ, интерпретацию и генерацию текстовых данных. Задачи могут быть разнообразными: классификация текста, машинный перевод, суммаризация, генерация текста и др. Особенностью является последовательная структура текста, где порядок слов играет ключевую роль.

Методы представления текста

Bag-of-Words: Один из базовых методов, когда текст представляется как вектор, где каждая компонента соответствует частоте появления слова в документе. Однако этот метод игнорирует порядок слов.

Word embeddings: Методы, такие как Word2Vec, GloVe, представляют слова в виде векторов фиксированной размерности, где семантически похожие слова имеют близкие векторы. Формально, функция f : слово $\to \mathbb{R}^d$ обучается так, чтобы:

cosine similarity $(f(w_i), f(w_j))$ была высокой, если слова w_i и w_j встречаются в схо

Контекстуальные представления: Современные модели, такие как BERT и GPT, используют трансформеры для построения контекстуальных эмбеддингов, где представление слова зависит от его окружения. Модель BERT обучается по принципу маскированного языкового моделирования, где часть слов заменяется специальным токеном [MASK], а

задача модели — предсказать пропущенные слова.

Модели и архитектуры

Рекуррентные нейронные сети (RNN) и LSTM: Ранее использовавшиеся для обработки последовательностей, где скрытое состояние обновляется с каждым новым словом:

$$z_t = \tanh\left(W_{xz}x_t + W_{zz}z_{t-1} + b_z\right).$$

Однако эти модели страдают от проблемы исчезающих градиентов при работе с длинными последовательностями.

Трансформеры: Основная инновация — механизм внимания (self-attention), который позволяет модели учитывать все входные позиции при формировании представления для каждого слова. Вычисление внимания производится по формуле:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

где Q (запросы), K (ключи) и V (значения) получаются из входных эмбеддингов с помощью обучаемых весовых матриц.

Примеры задач NLP

- **Классификация текста.** Например, определение тональности отзыва. Модель получает на вход векторное представление текста и предсказывает метку (положительная, отрицательная).
- **Машинный перевод.** Перевод предложений с одного языка на другой. Здесь применяются модели последовательного кодирования-декодирования, где кодировщик преобразует исходный текст в векторное представление, а декодировщик генерирует текст на целевом языке.
- **Генерация текста.** Задача, в которой модель, обученная на большом корпусе, генерирует новый текст, поддерживая грамматиче-

скую корректность и логическую связанность.

Математическая формализация NLP задач

Для классификации текста функция модели может выглядеть так:

$$\hat{y} = \text{softmax}(W\mathbf{h} + b),$$

где $\mathbf{h}$ — представление текста, полученное, например, посредством RNN, CNN или трансформера. Функция потерь — кросс-энтропия:

$$L = -\sum_{i=1}^{d} y_i \log \hat{y}_i.$$

Для задач генерации текста используется функция правдоподобия:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p(w_t \mid w_1, \ldots, w_{t-1}),$$

где w_t — слово на позиции t, а $p(w_t \mid w_1, \ldots, w_{t-1})$ вычисляется через механизм внимания и softmax.

Прогнозирование временных рядов

Общее описание задачи Прогнозирование временных рядов включает построение модели, способной предсказывать будущие значения последовательностей на основе их прошлых наблюдений. Примеры задач:

- Прогнозирование спроса на продукцию.
- Прогнозирование цен на финансовых рынках.
- Прогнозирование погодных условий.

Методы прогнозирования временных рядов Существуют как классические статистические методы, так и современные методы на основе нейронных сетей.

Классические методы:

- **Модель авторегрессии (AR):** Прогноз определяется как линейная комбинация прошлых наблюдений:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t,$$

  где c — константа, ϕ_i — коэффициенты, ϵ_t — шум.
- **Модель скользящего среднего (MA):** Использует прошлые ошибки для предсказания:

$$y_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}.$$

- **Модель ARIMA:** Объединяет авторегрессию, интегрирование и скользящее среднее, что позволяет моделировать тренды и сезонные компоненты.

Нейронные сети для временных рядов:

- **Многослойный персептрон (MLP):** Может использоваться для прогнозирования, если входной вектор составляется из значений временного ряда за предыдущие периоды.
- **Рекуррентные нейронные сети (RNN) и LSTM:** Способны моделировать временные зависимости за счёт использования скрытого состояния:

$$z_t = g\left(W_{xz} x_t + W_{zz} z_{t-1} + b_z\right),$$

  где x_t — значение временного ряда в момент времени t. LSTM дополнительно используют механизмы вентилей, чтобы избежать проблем исчезающих градиентов.
- **Сверточные нейронные сети (CNN):** Иногда применяются для извлечения признаков из временных рядов, особенно в сочетании

с RNN.

Математическая формализация и примеры Рассмотрим задачу прогнозирования временного ряда с использованием модели LSTM. Пусть наблюдения временного ряда заданы как $\{y_t\}_{t=1}^{T}$. Модель прогнозирует будущее значение $\hat{y}_{T+1}$ на основе предыдущих:

$$z_t = \text{LSTM}(x_t, z_{t-1}), \quad \hat{y}_t = f_{\text{out}}(z_t),$$

где $x_t = y_t$ (или же дополненный вектор признаков) и f_{out} — линейная функция:

$$\hat{y}_t = W_{\text{out}} z_t + b_{\text{out}}.$$

Функция потерь может быть определена как среднеквадратичная ошибка:

$$L = \frac{1}{T} \sum_{t=1}^{T} (y_t - \hat{y}_t)^2 .$$

Обучение модели происходит с помощью алгоритма обратного распространения ошибки во времени (BPTT).

0.4.2 Обучение без учителя

Обучение без учителя проводится без использования размеченных данных. Основная цель — найти скрытую структуру и закономерности в данных. Ниже приведены основные методы:

Карты Кохонена (Self-Organizing Maps)

Карты Кохонена (или *Self-Organizing Maps, SOM*) — это тип нейронных сетей без учителя, которые предназначены для отображения многомерных данных на двумерную решётку. Они выполняют кластеризацию и визуализацию, сохраняя топологические свойства исходного пространства признаков.

*Принцип работы

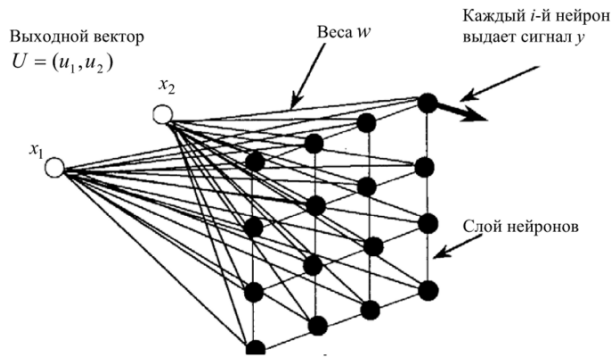

Рис. 2: Схематическое представление карты Кохонена: каждый входной вектор $x = (x_1, x_2)$ подключается ко всем нейронам двумерной решётки с соответствующими весами w. В процессе обучения активируется нейрон, весовой вектор которого ближе всего к входному (на основе евклидовой метрики), после чего его веса и веса соседей адаптируются к входу. Это обеспечивает кластеризацию данных с сохранением топологических связей.

Каждому узлу (нейрону) карты сопоставляется вектор весов той же размерности, что и входной вектор. В ходе обучения:

1. Выбирается случайный входной вектор.
2. Находится нейрон, чьи веса ближе всего к входу (*нейрон-победитель*).
3. Вес нейрона-победителя и его соседей корректируются: становятся ближе к входному вектору.

Таким образом, карта постепенно организуется так, чтобы похожие входные данные отображались в близких областях карты.

*Применения

- кластеризация данных;
- визуализация сложных пространств признаков;
- анализ текста, изображений, звука;
- выявление аномалий.

Кластеризация Алгоритмы кластеризации группируют данные на основе их сходства.

- **Алгоритм k-means:**
  - Инициализация: случайный выбор k центроидов.
  - Распределение данных: каждая точка x_i приписывается к ближайшему центроиду по евклидову расстоянию.
  - Обновление центроидов: каждый центроид пересчитывается как среднее значение всех точек, принадлежащих соответствующему кластеру.
  - Итерация: процесс повторяется до сходимости (когда центроиды перестают изменяться существенно).

Методы кластеризации Кластеризация разделяет данные на группы (кластеры), где объекты внутри кластера схожи между собой.

Алгоритм k-средних (k-means): Задача сводится к минимизации внутрикластерной дисперсии:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j} \|\mathbf{x} - \boldsymbol{\mu}_j\|^2,$$

где $\boldsymbol{\mu}_j$ — центр кластера C_j. Итеративно происходит:

1. Инициализация центров кластеров.
2. Назначение каждого объекта ближайшему центру.
3. Пересчёт центров как среднего значений объектов кластера.

Иерархическая кластеризация: Метод строит дендрограмму, объединяя данные в группы по принципу минимизации расстояния между кластерами. Существует агломеративный (снизу вверх) и дивизионный (сверху вниз) подходы. Критерии объединения могут быть различными (например, среднее расстояние, минимальное расстояние).

Методы понижения размерности

Эти методы используются для выявления внутренних структур в данных, позволяя уменьшить число переменных при сохранении значимой информации.

- **Principal Component Analysis (PCA):**
  - Цель PCA — найти ортогональное преобразование, которое проецирует данные на такое пространство, где первая компонента объясняет наибольшую дисперсию, вторая — вторую по величине, и так далее.
  - Вычисляется ковариационная матрица данных, затем её собственные значения и собственные векторы. Собственные векторы с наибольшими собственными значениями выбираются как главные компоненты.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE):**
  - t-SNE используется для визуализации данных в низкоразмерном пространстве (обычно 2D или 3D), сохраняя локальную структуру данных.
  - Алгоритм минимизирует расхождение между вероятностным распределением сходства точек в высокоразмерном пространстве и их представлением в низкоразмерном пространстве.

Автоэнкодеры

Автоэнкодеры — нейронные сети, обучающиеся копировать входные данные на выход, проходя через узкое скрытое представление.

- **Структура автоэнкодера:** Состоит из двух частей:

– **Энкодер:** Преобразует входной вектор x в скрытое представление h через нелинейное преобразование:

$$h = g(Wx + b).$$

– **Декодер:** Восстанавливает входной вектор из h:

$$\hat{x} = g'(W'h + b').$$

- **Цель обучения:** Минимизировать разницу между x и $\hat{x}$ (например, по квадратичной ошибке):

$$L(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2.$$

- **Пример:** При работе с изображениями автоэнкодер может научиться выделять основные признаки, такие как контуры и текстуры, позволяя затем использовать полученные скрытые представления для кластеризации или генерации новых изображений.

Вариационные автоэнкодеры (VAE) Вариационные автоэнкодеры расширяют классический автоэнкодер, обучаясь не только восстанавливать данные, но и моделировать их распределение.

- **Цель VAE:** Аппроксимировать распределение данных $p(x)$ с помощью скрытой переменной z. В отличие от обычного автоэнкодера, VAE вводят априорное распределение для z (обычно нормальное) и оптимизируют нижнюю оценку логарифмического правдоподобия.

- **Функция потерь VAE:** Состоит из двух частей — реконтрукционной ошибки и KL-дивергенции, измеряющей расхождение между апостериорным распределением $q(z|x)$ и априорным $p(z)$:

$$L(x, \hat{x}) = \frac{1}{2}\|x - \hat{x}\|^2 + D_{KL}\left(q(z|x)\|p(z)\right).$$

Подробное математическое обоснование VAE помогает понять, как с помощью обучения можно генерировать новые данные, похожие на исходные.

Мы формализуем задачу **обучения с подкреплением**, применяя идеи из теории динамических систем, а точнее как задачу оптимального управления не полностью известным **марковским процессом принятия решений**. Основную мысль можно сформулировать просто – требуется уловить наиболее важные аспекты реальной проблемы, стоящей перед обучающимся агентом, который взаимодействует во времени с окружающей средой для достижения некоторой цели. Обучающийся агент должен уметь в какой-то степени воспринимать состояние среды и предпринимать действия, изменяющие это состояние. У агента также должна быть цель или несколько целей, как-то связанных с состоянием окружающей среды. **Марковские процессы принятия решений** включают все три аспекта – восприятие, действие и цель – в простейшей возможной форме, не сводя, однако, ни один аспект к тривиальному. Любой метод, подходящий для решения таких задач, будет рассматриваться нами как метод **обучения с подкреплением**. При обучении с подкреплением агент взаимодействует с окружением и учится на основе обратной связи в виде вознаграждения.

- **Политика** $\pi(a|s)$ определяет вероятность выбора действия a в состоянии s.
- **Функция ценности** $V^\pi(s)$ или $Q^\pi(s,a)$ оценивает полезность состояния или пары состояние-действие.
- **Цель:** Максимизировать суммарное дисконтированное вознаграждение:
$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$
где γ — коэффициент дисконтирования.
- **Алгоритмы:** Примеры включают Q-обучение, SARSA, алгоритмы

градиентной политики и методы Actor-Critic. При использовании Q-обучения происходит обновление функции ценности согласно правилу:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right].$$

Эти методы позволяют агенту адаптироваться к динамичным условиям и оптимизировать своё поведение.

0.4.3 Динамическое программирование

- **ДП** — семейство алгоритмов для вычисления оптимальных стратегий в **марковских процессах принятия решений (МППР)**.
- Предполагает идеальную модель среды, что ограничивает практическое применение в **обучении с подкреплением** из-за вычислительной сложности.
- Теоретически важен: формирует основу для методов **обучения с подкреплением**, стремящихся достичь эффекта **ДП** с меньшими затратами и без идеальной модели.

Ключевые элементы МППР

- **Конечный МППР**: задается множествами состояний S, действий A, вознаграждений R.
- **Динамика среды** описывается вероятностями перехода $p(s', r|s, a)$.
- **Функции ценности** — инструмент для структурирования поиска стратегий.

Оценивание стратегии (Предсказание)

- Цель: вычислить функцию ценности $v_\pi(s)$ для произвольной стратегии π.

- **Алгоритм итеративного оценивания**:
  - На каждой итерации обновляются ценности всех состояний.
  - **Полное обновление**: учитывает все возможные переходы (математическое ожидание по всем следующим состояниям).
  - **Обновление на месте**: перезапись значений в одном массиве, что может ускорить сходимость.
- Сходимость гарантируется только в пределе, но на практике алгоритм останавливают при малых изменениях.

Улучшение стратегии

- **Жадная стратегия** π': выбирает действие, максимизирующее $q_\pi(s, a)$ (ожидаемую ценность).
- **Теорема об улучшении стратегии**: если π' жадная относительно v_π, то $v_\pi \geq v_{\pi'}$.
- **Стохастические стратегии**: допускают распределение вероятностей между действиями, максимизирующими ценность.

Асинхронное ДП

- Обновления производятся в произвольном порядке, без систематических проходов по состояниям.
- Преимущества:
  - **Гибкость**: можно интегрировать с реальным взаимодействием агента со средой.
  - **Эффективность** для больших пространств состояний.
- Условие сходимости: все состояния должны обновляться бесконечно часто.

Методы Монте-Карло

Введение в методы Монте-Карло

- **Определение**: Методы **МК** решают задачи обучения с подкреплением через усреднение выборочного дохода.
- **Область применения**: Эпизодические задачи (опыт делится на завершаемые эпизоды).
- **Особенности**:
  - Не требуют знания модели среды.
  - Обновления производятся поэпизодно, а не пошагово.
  - Основаны на полных доходах, полученных после посещения состояний или пар **состояние-действие**.

Предсказание методами Монте-Карло

- **Цель**: Оценка функции ценности состояний $v_\pi(s)$ для заданной стратегии π.
- **Методы**:
  - **МК первого посещения**: усредняет доходы, полученные после первого посещения состояния s в эпизоде.
  - **МК всех посещений**: усредняет доходы после всех посещений s.
- **Сходимость**: Оба метода сходятся к $v_\pi(s)$ при бесконечном числе посещений.
- **Преимущества**:
  - Метод первого посещения проще и имеет меньшую дисперсию.
  - Метод всех посещений квадратично сходится, но сложнее в реализации.

Обучение на основе временных различий

Ключевые термины

- **TD-обучение**: Инкрементальное обновление на основе временных различий.
- **Бутстрэппинг**: Использование текущих оценок для обновления.
- **Sarsa/Q-обучение**: Алгоритмы управления с единой/разделенной стратегией.
- **Смещение максимизации**: Артефакт использования максимума оценок.
- **Послесостояния**: Состояния после действия с известной динамикой.

Введение в TD-обучение

- **TD-обучение** (обучение на основе временных различий) объединяет идеи методов **Монте-Карло (МК)** и **динамического программирования (ДП)**:
  - Как **МК**: обучается на опыте без модели окружения.
  - Как **ДП**: использует **бутстрэппинг** (обновляет оценки на основе других оценок, не дожидаясь конечного результата).
- **Ключевое отличие от МК**:
  - **МК** ждет конца эпизода для обновления оценки (цель — полный доход).
  - **TD** обновляет оценки после каждого шага, используя наблюдаемое вознаграждение R_{t+1} и оценку следующего состояния $V(S_{t+1})$.
- **Пример**: **TD(0)** (одношаговый TD) — базовая версия, где обновление происходит сразу после перехода.

0.5 Методы обучения нейронных сетей

0.5.1 Функции потерь и их назначение

Функция потерь (loss function) измеряет «стоимость» ошибки модели. Она называется так, поскольку определяет потери, которые несёт система из-за несовпадения предсказаний с истинными значениями. Рассмотрим основные типы:

- **Квадратичная ошибка (MSE):**

$$L(f(x), y) = \frac{1}{2}\big(y - f(x)\big)^2.$$

  Здесь ошибка определяется как квадрат разницы между истинным и предсказанным значениями, что обеспечивает сильное наказание за большие ошибки. Квадратичная функция гладкая и дифференцируемая, что упрощает оптимизацию.

- **Перекрестная энтропия:**

$$L(f(x), y) = -\sum_{k=1}^{K} \delta_{yk} \log f_k(x).$$

  Эта функция измеряет расстояние между истинным распределением классов и предсказанным, опираясь на концепцию энтропии из теории информации. Она особенно полезна, когда выход модели интерпретируется как вероятность.

- **Hinge Loss** ("потери с зазором"используется в задачах двухклассовой классификации, особенно в опорных векторах): Применяется в задачах, где важна максимизация зазора между классами, как в SVM.

0.5.2 Алгоритм градиентного спуска

Общее описание

Алгоритм градиентного спуска является базовым методом оптимизации, используемым для минимизации функции потерь $L(w)$ по параметрам w. Идея заключается в том, чтобы итеративно обновлять параметры в направлении, противоположном градиенту функции потерь.

Математическая постановка

Пусть функция потерь $L(w)$ зависит от параметров $w \in \mathbb{R}^n$. Тогда шаг обновления параметров определяется формулой:

$$w \leftarrow w - \alpha \nabla_w L(w),$$

где:

- $\alpha > 0$ — скорость обучения (learning rate);
- $\nabla_w L(w)$ — градиент функции потерь по параметрам w.

Пример: линейная регрессия

Рассмотрим модель линейной регрессии:

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_n x_n = \mathbf{w}^\top \mathbf{x} + w_0.$$

Функция потерь (среднеквадратичная ошибка) имеет вид:

$$L(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right)^2,$$

где N — число обучающих примеров.

Вычислим градиент по параметру w_j:

$$\frac{\partial L}{\partial w_j} = -\frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}.$$

Обновление параметров:

$$w_j \leftarrow w_j + \alpha \cdot \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}.$$

Этот процесс повторяется до сходимости функции потерь.

0.5.3 Обратное распространение ошибки (Backpropagation)

Общее описание

Обратное распространение ошибки (backpropagation) — алгоритм вычисления градиентов для многослойных нейронных сетей. Он основан на правиле цепочки и позволяет эффективно передавать ошибку от выходного слоя к каждому предыдущему слою для обновления весов.

Математическая постановка

Пусть нейронная сеть имеет L слоёв, где на l-м слое вычисляется:

$$\mathbf{z}^{(l)} = W^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)},$$

$$\mathbf{h}^{(l)} = g^{(l)}\big(\mathbf{z}^{(l)}\big),$$

где $\mathbf{h}^{(0)} = \mathbf{x}$ — входные данные, а $g^{(l)}(\cdot)$ — функция активации.

Если функция потерь L определяется на выходном слое $\mathbf{h}^{(L)}$, то для обратного распространения градиентов используется правило цепочки.

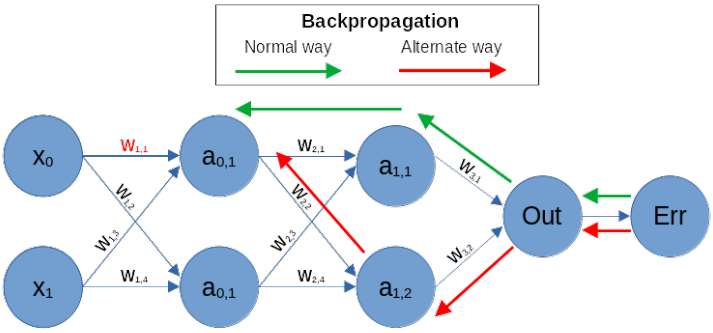

Рис. 3: Схема прямого и обратного распространения в нейронной сети: прямой проход (синие стрелки), обратное распространение ошибки (зелёные — стандартный путь, красные — альтернативный путь).

Градиент по параметрам $W^{(l)}$ вычисляется следующим образом:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \mathbf{z}^{(l)}} \left(\mathbf{h}^{(l-1)}\right)^\top,$$

где

$$\frac{\partial L}{\partial \mathbf{z}^{(l)}} = \left(W^{(l+1)}\right)^\top \frac{\partial L}{\partial \mathbf{z}^{(l+1)}} \odot g'^{(l)}\left(\mathbf{z}^{(l)}\right).$$

Здесь $\odot$ обозначает поэлементное умножение, а $g'^{(l)}$ — производная функции активации.

Пример: нейрон с ReLU

Нейрон с ReLU (Rectified Linear Unit) ускоряет обучение по сравнению с сигмоидами и tanh, так как: не вызывает затухание градиентов (градиент равен 1 для положительных значений); работает быстрее из-за простоты вычислений; способствует разреженности активаций (многие нейроны "молчат т.е. дают 0), что улучшает обобщение. ReLU позволяет строить глубокие нейросети, которые обучаются быстро и эффективно. Это стандартная активация в скрытых слоях современных ИНС. Для нейрона с функцией активации ReLU, где

$$g(z) = \max(0, z),$$

производная имеет вид:

$$g'(z) = \begin{cases} 1, & z > 0, \\ 0, & z \leq 0. \end{cases}$$

Если на выходном слое применяется кросс-энтропия, то ошибка $\frac{\partial L}{\partial h^{(L)}}$ вычисляется напрямую, а затем распространяется назад по сети с помощью описанного правила цепочки.

0.5.4 Обратное распространение ошибки во времени (BPTT) для RNN

Общее описание

Обратное распространение ошибки во времени (Backpropagation Through Time, BPTT) используется для обучения рекуррентных нейронных сетей (RNN), где скрытое состояние зависит от предыдущих состояний. При BPTT сеть «разворачивается» во времени, и градиенты вычисляются для каждого временного шага, а затем суммируются.

Математическая постановка

Пусть RNN описывается следующей динамикой:

$$z_t = g\Big(W_{xz}x_t + W_{zz}z_{t-1} + b\Big), \quad t = 1, \ldots, T,$$

а выходной сигнал:

$$\hat{y}_t = f\big(z_t\big).$$

Функция потерь определяется как суммарная ошибка по всем временным шагам:

$$L = \frac{1}{T}\sum_{t=1}^{T} L_t\big(\hat{y}_t, y_t\big).$$

Градиент по параметру W_{zz} вычисляется с учётом вклада от каждого временного шага:

$$\frac{\partial L}{\partial W_{zz}} = \sum_{t=1}^{T} \frac{\partial L}{\partial z_t} \frac{\partial z_t}{\partial W_{zz}}.$$

При этом

$$\frac{\partial z_t}{\partial W_{zz}} = g'\Big(W_{xz}x_t + W_{zz}z_{t-1} + b\Big) \cdot z_{t-1},$$

а градиент $\frac{\partial L}{\partial z_t}$ вычисляется рекурсивно:

$$\frac{\partial L}{\partial z_t} = \frac{\partial L_t}{\partial z_t} + (W_{zz})^\top \left[\frac{\partial L}{\partial z_{t+1}} \odot g'\Big(W_{xz}x_{t+1} + W_{zz}z_t + b\Big)\right].$$

Таким образом, ошибка от более поздних временных шагов «обратно распространяется» к предыдущим, что позволяет учитывать влияние прошлого на текущий выход.

Применение BPTT

Временной ряд — это последовательность значений, измеренных через равные промежутки времени. Примеры временных рядов: температура воздуха по дням; курс акций по минутам; трафик на сайте по часам; потребление электроэнергии по неделям. Такие данные естественным образом упорядочены во времени, и их основная особенность — зависимость текущего значения от предыдущих. Именно эту зависимость и пытается уловить RNN — рекуррентная нейронная сеть — чтобы предсказать будущее значение. Рассмотрим простую RNN для прогнозирования временного ряда. Пусть входное значение x_t равно предыдущему наблюдению, а модель предсказывает $\hat{y}_t = z_t$. Тогда:

$$z_t = \tanh\Big(W_{xz}x_t + W_{zz}z_{t-1} + b\Big).$$

Функция потерь может быть выбрана как среднеквадратичная ошибка:

$$L = \frac{1}{T} \sum_{t=1}^{T} \big(y_t - z_t\big)^2.$$

Используя BPTT, вычисляются градиенты по W_{zz} и другим параметрам с учётом влияния предыдущих состояний. Этот метод позволяет модели учитывать длительные зависимости, хотя и требует специальной техники для борьбы с исчезающими или взрывными градиентами.

0.5.5 Методы нормализации входных данных и регуляризации параметров модели

Нормализация необходима входным данным для того, чтобы значения имели единый масштаб и распределение (например, с нулевым средним и единичной дисперсией). Это ускоряет и стабилизирует обучение, особенно при использовании градиентного спуска.

Регуляризация параметров модели делается с целью уменьшения переобучения, когда модель слишком точно "запоминает"обучающую выборку, но плохо обобщает на новые данные. Используются такие методы, как L1 и L2-регуляризация (добавление штрафов к функции потерь), а также Dropout (случайное "отключение"нейронов во время обучения).

Пакетная нормализация (Batch Normalization)

Пакетная нормализация применяется для стабилизации и ускорения обучения. Пусть для активаций $z_1, z_2, \ldots, z_m$ в мини-пакете вычисляются среднее и дисперсия:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} z_i, \quad \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (z_i - \mu)^2.$$

Нормализованные значения:

$$\hat{z}_i = \frac{z_i - \mu}{\sqrt{\sigma^2 + \epsilon}},$$

а затем применяется масштабирование и смещение:

$$z_i^{\text{out}} = \gamma\, \hat{z}_i + \beta,$$

где γ и β — обучаемые параметры, а ϵ — небольшая константа для предотвращения деления на ноль.

Регуляризация моделей

Чтобы избежать переобучения и повысить обобщающую способность модели, используются следующие методы:

Сокращение веса (Weight Decay): К функции потерь добавляется штраф за большие значения весов:

$$L_{\text{reg}} = L + \lambda \sum_i w_i^2,$$

где λ регулирует степень регуляризации.

Метод исключения (Dropout): Во время обучения случайным образом отключаются (обнуляются) нейроны с заданной вероятностью. Это заставляет сеть учиться работать с неполной информацией и снижает зависимость от отдельных узлов.

0.5.6 Рекуррентные нейронные сети и LSTM

Базовая архитектура RNN

В рекуррентной нейронной сети (RNN) скрытое состояние — это внутренняя память, которая накапливает информацию о предыдущих элементах последовательности. Рекуррентные нейронные сети предназначены для обработки последовательных данных. Уравнение обновления скрытого состояния показывает, как эта память изменяется на каждом шаге:

$$z_t = g\left(W_{xz}x_t + W_{zz}z_{t-1} + b_z\right),$$

где x_t — вход в момент времени t, z_t — скрытое состояние, а g — нелинейная функция (например, `tanh`).

Обратное распространение во времени (BPTT)

Для вычисления градиентов в RNN применяется метод BPTT, позволяющий передавать ошибку от выходного слоя к каждому предыдущему состоянию:

$$\frac{\partial L}{\partial W_{zz}} = \sum_{t=1}^{T} \frac{\partial L}{\partial z_t} \cdot g'\Big(W_{xz}x_t + W_{zz}z_{t-1} + b_z\Big) \cdot z_{t-1}.$$

Такой подход позволяет учитывать влияние прошлых состояний, но требует осторожного выбора гиперпараметров для избежания исчезающих или взрывных градиентов.

Архитектура LSTM (Long Short-Term Memory)

LSTM были разработаны для решения проблем длительной зависимости в RNN. Основные компоненты LSTM включают:

- **Вентиль забывания:** определяет, какая часть информации из

предыдущей ячейки памяти сохраняется:

$$f_t = \sigma\left(W_f \cdot [z_{t-1}, x_t] + b_f\right).$$

- **Входной вентиль:** регулирует поступление новой информации:

$$i_t = \sigma\left(W_i \cdot [z_{t-1}, x_t] + b_i\right).$$

- **Кандидат обновления памяти:**

$$\tilde{c}_t = \tanh\left(W_c \cdot [z_{t-1}, x_t] + b_c\right).$$

- **Обновление ячейки памяти:**

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

где $\odot$ обозначает поэлементное умножение.

- **Выходной вентиль:**

$$o_t = \sigma\left(W_o \cdot [z_{t-1}, x_t] + b_o\right),$$

и окончательный выход:

$$z_t = o_t \odot \tanh(c_t).$$

Такая архитектура позволяет сохранять информацию на протяжении длительных временных интервалов и эффективно решать задачи, связанные с обработкой последовательностей, например, в системах машинного перевода или распознавания речи.

0.6 Современные архитектуры нейронных сетей

0.6.1 Сверточные нейронные сети (CNN)

CNN применяются для обработки изображений. Их ключевые компоненты:

- **Сверточные слои:** На каждом слое происходит свёртка входного изображения с набором обучаемых фильтров, что позволяет выделять пространственные признаки, такие как края, текстуры и формы.
- **Pooling-слои:** Снижают размерность признаков за счёт агрегации (например, максимальное или среднее значение) по небольшим областям.
- **Полносвязные слои:** Завершают процесс классификации, преобразуя извлечённые признаки в конечный ответ.

Принцип работы сверточного слоя. Каждое обучаемое ядро (фильтр) K скользит по входному изображению X с шагом (stride) s и на каждой позиции вычисляет свёртку:

$$Y_{i,j} = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} K_{u,v} \cdot X_{i+u,j+v}$$

Это соответствует вычислению скалярного произведения окна входа с ядром. Каждый фильтр фокусируется на определённом признаке (градиент, текстура и т.д.), формируя карту признаков (feature map).

Интуиция: на ранних слоях фильтры извлекают базовые паттерны (границы, углы), а на более глубоких — более абстрактные признаки (формы, части объектов).

Пример: При размере входа 32×32, ядре 5×5, без padding и шаге 1, размер выхода будет 28×28.

Обучение: Фильтры обучаются с помощью обратного распространения ошибки (backpropagation), как и в обычных нейронных сетях.

0.6.2 Рекуррентные нейронные сети (RNN), LSTM и GRU

RNN используются для обработки последовательностей, где важна информация о порядке элементов:

- **Классические RNN:** Каждое состояние h_t вычисляется с учетом текущего входа x_t и предыдущего состояния h_{t-1}:

$$h_t = g(W_{xh}x_t + W_{hh}h_{t-1} + b_h).$$

- **LSTM:** Используют ячейки памяти и ворота (входной, забывания и выходной), чтобы эффективно запоминать долгосрочную информацию:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i), \quad f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f),$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o), \quad \tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c),$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad h_t = o_t \odot \tanh(c_t).$$

- **GRU:** Упрощают архитектуру LSTM, объединяя ворота забывания и обновления:

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z), \quad r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r),$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h), \quad h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t.$$

Интуиция: RNN поддерживает скрытое состояние h_t, которое обобщает всю историю предыдущих входов. Однако простые RNN плохо работают с длинными зависимостями из-за затухающего градиента.

LSTM: работа ворот. Каждое «ворота» контролирует поток информации:

- **Забывание (f_t):** какие части старого состояния c_{t-1} нужно забыть.
- **Вход (i_t, $\tilde{c}_t$):** какие новые данные добавить.
- **Выход (o_t):** какую часть состояния c_t отдать наружу как h_t.

Благодаря этим операциям, LSTM может запоминать важную информацию на протяжении десятков или сотен шагов, избегая проблемы затухающего градиента.

GRU: сравнение. GRU проще: объединяет забывание и обновление в одно «обновляющее» ворота z_t, и не имеет отдельного состояния c_t. Поэтому работает быстрее, но может быть менее гибким.

Обучение: Все веса (в матрицах W_{x*}, W_{h*}) обучаются градиентным спуском с использованием обратного распространения через время (BPTT).

0.6.3 Генеративно-состязательные сети (GAN)

GAN состоят из двух моделей, обучающихся в противостоянии:

- **Генератор** G генерирует данные, имитирующие реальные, преобразуя случайный шум z в синтетические данные $G(z)$.
- **Дискриминатор** D оценивает, насколько сгенерированные данные похожи на реальные.

Оптимизационная задача GAN формулируется так:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\big[\log D(x)\big] + \mathbb{E}_{z\sim p_z(z)}\big[\log\big(1 - D(G(z))\big)\big].$$

Подробное описание процесса обучения GAN, возникающих проблем (например, mode collapse) и методов стабилизации раскрывается в современных исследованиях.

Интуиция: Генератор G пытается «обмануть» дискриминатор D, создавая фальшивые данные, похожие на реальные. Дискриминатор стремится отличить настоящие данные x от синтетических $G(z)$.

Обучение:

- Дискриминатор обновляется, чтобы максимизировать вероятность правильной классификации реальных и фейковых данных.
- Генератор обновляется, чтобы минимизировать способность дискриминатора различать фейки.

Обновление градиентов происходит поочерёдно: сначала D, затем G.

Проблемы:

- **Mode collapse:** генератор выдает однотипные данные.
- **Неустойчивость:** градиенты могут исчезать или взрываться.

Решения:

- Использование Wasserstein-GAN (с расстоянием Вассерштейна).
- Введение регуляризации (например, gradient penalty).

0.6.4 Трансформеры

Трансформеры применяются для обработки последовательностей с помощью механизма self-attention.

- **Self-attention:** Вычисляет взаимосвязи между всеми элементами входной последовательности независимо от их расстояния, позволяя выделить важные зависимости.
- **Encoder-Decoder:** Энкодер преобразует входную последовательность в скрытое представление, а декодер генерирует выходное представление.
- **Позиционные кодировки:** Дополняют входные данные информацией о порядке элементов.

Основное выражение внимания выглядит следующим образом:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

где Q — матрица запросов, K — ключей, V — значений, а d_k — размерность ключей.

Подробный разбор Attention. Каждое слово преобразуется в три вектора:

- Q (запрос) — кого я ищу?
- K (ключ) — что я предлагаю?
- V (значение) — какую информацию я несу?

Затем вычисляется сходство между запросом q_i и ключами k_j:

$$\alpha_{ij} = \frac{\exp(q_i \cdot k_j/\sqrt{d_k})}{\sum_{j'} \exp(q_i \cdot k_{j'}/\sqrt{d_k})}$$

Итоговое представление слова:

$$y_i = \sum_j \alpha_{ij} v_j$$

Это позволяет «посмотреть» на все слова и выделить важные.

Multi-head Attention: Несколько голов внимания работают параллельно. Каждая фокусируется на разных аспектах (например, синтаксис, семантика). Выходы объединяются:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_h) W^O$$

Feed-Forward блок: После attention-преобразований на каждом токене применяется одинаковый двухслойный MLP:

$$\text{FFN}(x) = \max(0, xW_1 + b_1) W_2 + b_2$$

Нормализация и остаточные связи: Каждый блок окружён нормализацией и skip-связью:

$$x' = \text{LayerNorm}(x + \text{Sublayer}(x))$$

Это стабилизирует градиенты и ускоряет обучение.

Позиционные кодировки: Т.к. attention не учитывает порядок слов, позиционные кодировки добавляются к входным эмбеддингам:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

0.7 Практические примеры применения ИНС и современные инструменты

0.7.1 Примеры практических задач

Распознавание образов

Описание задачи: Распознавание образов предполагает автоматическую идентификацию объектов на изображениях. Ключевая цель — построение модели, способной классифицировать изображения, выделять объекты и даже выполнять локализацию (определять, где на изображении находится объект).

Методы и архитектуры:

- **Сверточные нейронные сети (CNN).** Эти сети используют сверточные слои, которые применяют набор фильтров для выделения локальных признаков. Например, на первых слоях CNN обнаруживаются границы, линии и текстуры, а на последующих слоях происходит построение всё более абстрактных представлений.
- **Пулинг (Pooling).** После сверточных слоёв применяются операции пулинга (например, max-pooling), которые уменьшают размерность признакового пространства и повышают устойчивость к небольшим смещениям объектов.

Пример: Для распознавания лиц можно использовать модель, которая на вход принимает изображение, предварительно нормализованное и масштабированное до фиксированного размера. Затем сверточные слои выделяют ключевые признаки лица (например, глаза, нос, рот), а полносвязные слои на выходе классифицируют изображение по идентичности. Такие подходы описаны в работах Рассела и Норвега, где приводятся примеры применения CNN для распознавания объектов, и в книге Хайкина, где подробно рассматриваются алгоритмы обучения сверточных

сетей.

Обработка естественного языка (NLP)

Описание задачи: Обработка естественного языка включает анализ, интерпретацию и генерацию текстовой информации. Задачи могут варьироваться от классификации текстов и анализа тональности до машинного перевода и генерации диалогов.

Методы представления текста:

- **Бэг-оф-слов (Bag-of-Words) и TF-IDF.** Классические подходы, при которых текст представляется в виде вектора, где каждое измерение соответствует частоте появления слова или взвешенной частоте.
- **Word Embeddings (Word2Vec, GloVe).** Слова кодируются в виде плотных векторов фиксированной размерности, что позволяет учитывать семантические отношения между словами.
- **Контекстуальные модели (BERT, GPT).** Современные трансформерные архитектуры создают представление слов с учётом их контекста, что значительно повышает качество решения задач, таких как машинный перевод или генерация текста.

Пример: Для задачи машинного перевода используется архитектура «кодировщик-декодировщик», где кодировщик преобразует входное предложение на одном языке в контекстное представление, а декодировщик генерирует перевод на целевом языке. Рассмотренные в работах Рассела и Норвега модели трансформеров демонстрируют высокую точность перевода даже для длинных предложений.

Регрессия и аппроксимация временных рядов

Описание задачи: Регрессия и аппроксимация применяются для прогнозирования непрерывных величин, таких как финансовые показатели, температура или динамика физических процессов. Здесь модель обучается находить зависимости между входными признаками и выходным значением.

Методы:

- **Линейная регрессия.** Простейшая модель, которая предполагает линейную зависимость между входными переменными и целевой величиной. Метод минимизации среднеквадратичной ошибки позволяет находить оптимальные параметры.
- **Нейронные сети (MLP, RNN, LSTM).** Более сложные модели, способные моделировать нелинейные зависимости и учитывать временную динамику. Рекуррентные нейронные сети (RNN) и их усовершенствованные варианты, такие как LSTM, особенно полезны для анализа временных рядов, так как они сохраняют информацию о предыдущих наблюдениях.

Пример: В задачах прогнозирования цен на акции часто используется модель LSTM, которая обучается на исторических данных, чтобы предсказывать будущее значение. В книге Хайкина подробно описаны методы аппроксимации сложных функций с помощью нейронных сетей, а в работах Рассела и Норвега показаны примеры применения подобных моделей в финансовом моделировании.

Управление и оптимизация

Описание задачи: В задачах автономного управления и оптимизации производственных процессов агент на основе ИНС принимает решения, корректируя свою стратегию в реальном времени на основе входных

данных. Здесь важно не только классифицировать или аппроксимировать информацию, но и принимать оптимальные решения в динамической среде.

Методы:

- **Обучение с подкреплением (Reinforcement Learning).** Агент взаимодействует со средой, получая обратную связь в виде вознаграждения. На основе этой обратной связи обучается политика, максимизирующая суммарное вознаграждение.
- **Гибридные подходы.** Комбинация обучения с учителем и обучения с подкреплением позволяет сначала обучить модель на размеченных данных, а затем дообучать её в реальных условиях, корректируя поведение агента.

Пример: В робототехнике ИНС используются для автономного управления роботами. Агент обучается на симуляциях и реальных данных для выполнения задач, таких как навигация, захват объектов или управление производственными линиями. Подходы, описанные в работах Рассела и Норвега, показывают, как интегрировать методы обучения с подкреплением в системы автономного управления, а Хайкин даёт глубокое понимание структуры нейронных сетей, применяемых для оптимизации сложных процессов.

0.7.2 Современные инструменты и фреймворки

Для разработки ИНС используются популярные инструменты:

- **TensorFlow** и **PyTorch** — библиотеки для построения вычислительных графов и автоматического дифференцирования.
- **Keras** — высокоуровневая оболочка для быстрого прототипирования.

- **Scikit-learn** — набор инструментов для классических методов машинного обучения.
- **OpenCV** — для обработки изображений и интеграции с нейросетевыми моделями.

0.7.3 Применение ИНС агентом

Современные интеллектуальные агенты, построенные на основе ИНС, способны выполнять комплекс задач в различных областях:

- **Классификация и распознавание.** Агент обрабатывает входные данные (изображения, текст, аудио) и определяет, к какому классу они принадлежат. Например, система распознавания лиц в системах безопасности или классификатор текста в социальных сетях.
- **Аппроксимация сложных функций.** Нейронные сети обучаются моделировать зависимости между входными и выходными параметрами. Это используется для прогнозирования динамики систем, таких как финансовые рынки или технические процессы.
- **Регрессия.** Прогнозирование непрерывных величин (температуры, давления, расхода ресурсов) с высокой точностью.
- **Контроль и оптимизация.** В системах автономного управления агент, обученный на основе ИНС, способен адаптивно корректировать стратегии управления, оптимизировать производственные процессы и принимать решения в режиме реального времени.

0.8 Ограничения нейросетевого подхода и перспективы развития

0.8.1 Требования к данным и вычислительным ресурсам

Глубокие нейронные сети требуют больших объёмов данных для эффективного обучения и значительных вычислительных ресурсов (GPU, TPU), что может ограничивать их применение в некоторых узкоспециализированных задачах.

0.8.2 Интерпретируемость моделей

Современные ИНС часто являются «черными ящиками», что затрудняет понимание причин принятия решений. Это особенно важно для приложений, требующих объяснимости, например, в медицине или юриспруденции.

0.8.3 Переобучение и обобщающая способность

Недостаток данных или некорректная настройка параметров может привести к переобучению, когда модель слишком точно запоминает обучающую выборку и теряет способность обобщать знания на новые данные. Применяются методы регуляризации, такие как dropout, L2-регуляризация и ранняя остановка, для снижения этого риска.

0.8.4 Проблемы градиентного спуска

При обучении глубоких сетей могут возникать проблемы исчезающих и взрывающихся градиентов. Для их решения используются функции активации типа ReLU, нормализация (batch normalization) и адаптивные методы оптимизации.

0.8.5 Перспективы развития

Будущие исследования направлены на:

- Создание гибридных моделей, объединяющих нейросетевые и символические подходы для повышения интерпретируемости.
- Разработку методов обучения, требующих меньше данных (transfer learning, few-shot learning).
- Повышение устойчивости моделей к динамическим изменениям внешней среды.
- Интеграцию ИНС в комплексные системы автономного управления, где требуется не только точное распознавание, но и обоснованное принятие решений.

[1] Рассел С., Норвиг П. *Искусственный интеллект: современный подход. 4-е изд., том 1. Решение проблем: знания и рассуждения.* СПб.: Диалектика, 2021.

[2] Рассел С., Норвиг П. *Искусственный интеллект: современный подход. 4-е изд., том 3. Обучение, восприятие и действие.* СПб.: Диалектика, 2022.

[3] Хайкин С. *Нейронные сети: полный курс, 2-е изд.* М.: Вильямс, 2006.

[4] «Искусственные нейронные сети» [Электронный ресурс]. `http://bigor.bmstu.ru/?cnt/?doc=NN/base.cou`.