university
Тюменский
индустриальный
университет

ОСНОВЫ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Учебное пособие

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»

ОСНОВЫ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Учебное пособие

Тюмень
ТИУ
2024

УДК 004.8
ББК 16.6
О -75

Авторы:
А. С. Гаваев, И. В. Лысенко, Д. А. Чайников, А. С. Губенко

Рецензенты:
доктор технических наук,
профессор кафедры автоматизации производственных процессов
Курганского государственного университета
В. Е. Овсянников;
кандидат геолого-минералогических наук, доцент кафедры
математики и прикладных информационных технологий ТИУ
С. А. Чунихин

Основы систем искусственного интеллекта: учебное пособие /
О -75 А. С. Гаваев, И. В. Лысенко, Д. А. Чайников, А. С. Губенко. –
Тюмень: ТИУ, 2024. – 88 с. – Текст: непосредственный.
ISBN 978-5-9961-3402-1

В учебном пособии представлены понятия, области применения и математические основы искусственного интеллекта, виды и алгоритмы машинного обучения. Особое внимание уделено построению систем искусственного интеллекта на основе нейронных сетей. Изложены основы обучения нейронных сетей с учителем, без учителя и обучения с подкреплением.

Учебное пособие предназначено для обучающихся высших учебных заведений направлений подготовки 15.04.01 «Машиностроение»; 15.03.01 «Машиностроение»; 27.03.05 «Инноватика». Учебное пособие или его части также могут использоваться для формирования профессиональных компетенций обучающихся высших учебных заведений других направлений подготовки, а также рекомендуется для повышения квалификации научных сотрудников, выполняющих исследования в сфере создания и использования систем искусственного интеллекта в различных областях производства.

УДК : 004.8
ББК : 16.6

ISBN 978-5-9961-3402-1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие систем искусственного интеллекта (ИИ) и связанных с ним технологий, сопровождается значительным ростом внедрения технических решений на основе искусственного интеллекта. Разработка и внедрение прикладных технологических решений с использованием систем искусственного интеллекта существенного повышает производительность человека в различных сферах деятельности.

Национальную стратегию развития искусственного интеллекта на период до 2030 года утверждает Указ Президента Российской Федерации «О развитии искусственного интеллекта в Российской Федерации» от 10 октября 2019 г. № 490. Применение стратегии направлено на развитие и внедрение систем искусственного интеллекта во всех областях производственной деятельности и бытовой жизни населения страны.

Усвоение математических основ в области изучения больших данных с применением систем искусственного интеллекта являются обязательным условием для развития ИИ, проведения научных исследований и совершенствования навыков при подготовке кадров.

Формирование знаний о функционирований систем искусственного интеллекта включает: изучение математических методов исследования работы систем; рассмотрение вопросов математической статистики и теории вероятности; умение применять методы линейной алгебры и матричные операции и т.д.

Искусственный интеллект как комплекс компьютерных систем имитирующих когнитивные функции человека является междисциплинарной наукой. В связи с этим для создания систем ИИ необходимы специалисты из различных сфер деятельности: инженеры, программисты, психологи, экономисты, математики, нейрофизиологи и т.д. Методы систем искусственного интеллекта многообразны, используются и адаптируются в зависимости от поставленной задачи, заимствуются и объединяются из разных наук.

Направлением развития ИИ является моделирование естественного интеллекта человека с имитацией процессов рассуждения, логических выводов и принятия решений.

Глубокое обучение является важной актуальной областью искусственного интеллекта. Развитием данного направления является разработка прогрессивных математических методов моделирования, прогнозирования поведения моделей и эффективное управление ими.

В настоящее время в исследованиях систем искусственного интеллекта существенное внимание вызывает изучение нейронных сетей с использованием математических методов.

1. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ОСНОВЫ МАШИННОГО ОБУЧЕНИЯ

Основной целью данного раздела является изучить основные понятия и базовые концепции искусственного интеллекта и машинного обучения на примерах различных видов регрессионного анализа.

1.1. Область применения и основные понятия

С математической точки зрения существуют два основных подхода к моделированию: детерминированное моделирование и стохастическое моделирование (рис. 1.1).

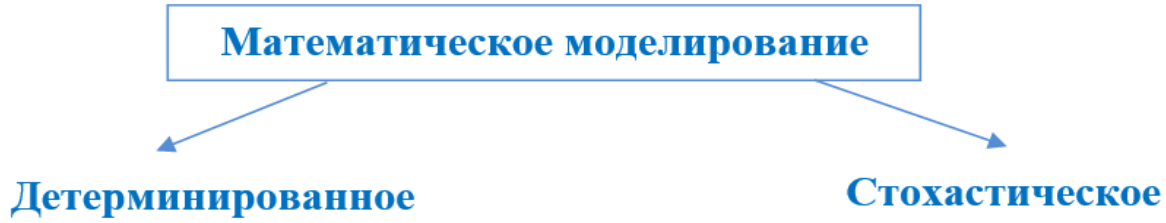


Рисунок 1.1 – Основные подходы математического моделирования

В первом случае, связь между входными и выходными данными известна и определяется какими-либо физическими законами. Простым примером такой связи может служить задача из курса физики в которой рассматривается движение ядра под воздействием силы тяжести в вертикальном направлении, выпущенного со стартовой скоростью. С точки зрения детерминированного моделирования, данная задача имеет известное, хорошо изученное и простое решение, основанное на законе движения: ускорение равно g и зная стартовую скорость и положение, можно определить положение ядра после первого дифференцирования закона скорости и закона движения после второго интегрирования.

Формулы для решения этой задачи приведены ниже:

$$\frac{d^2x}{dt^2} = -g\,. \tag{1.1}$$

при $V(0)=V_0$; $x(0)=x_0$; $a=-g$.

$$V=-gt+V_0. \tag{1.2}$$

$$X = -\frac{gt^2}{2} + V_0 t + x_0\,. \tag{1.3}$$

Первое интегрирование дает скорость представлено в виде уравнение 1.2, второе – положение объекта (уравнение 1.3). Таким образом, зная начальные условия, в любой момент времени можно определить все параметры движения тела. С другой стороны, ту же задачу можно решить с помощью стохастического подхода. Если имеется возможность каким-то образом измерить положение тела в любой момент времени, эти данные могут быть аппроксимированы (рис. 1.2).

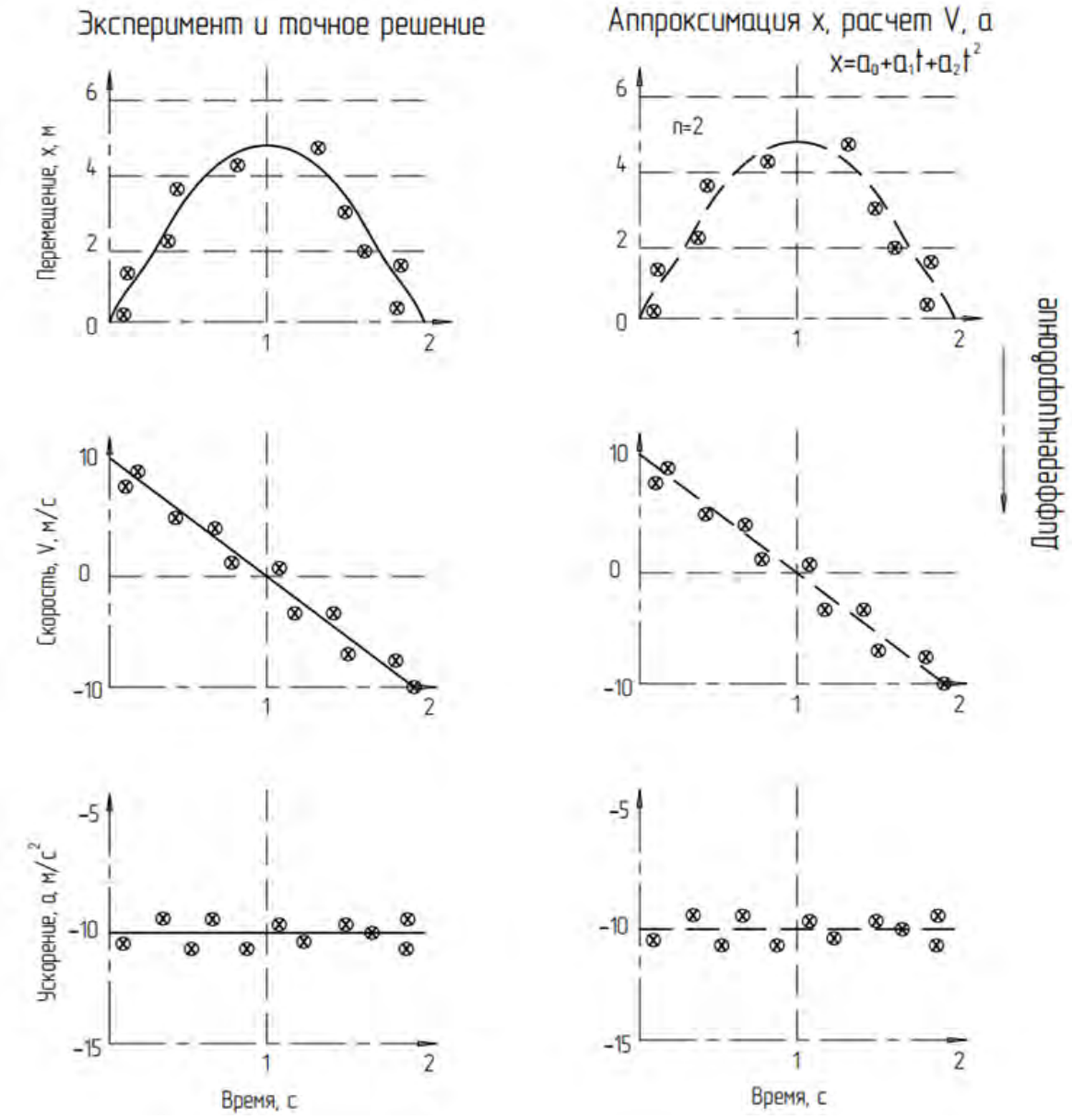


Рисунок 1.2 – Стохастический подход. Аппроксимирование

В представленном случае на рисунке 1.2 видно, что эти данные могут быть аппроксимированы параболой и можно записать в виде функции $x = Q_0 + Q_1t + Q_2t^2$. Далее можно продифференцировать данную функцию один и два раза, чтобы получить скорость и ускорение соответственно.

Альтернативный вариант – это провести эксперимент для определения ускорения в каждый момент времени и затем аппроксимировать функцию ускорения (рис. 1.3).

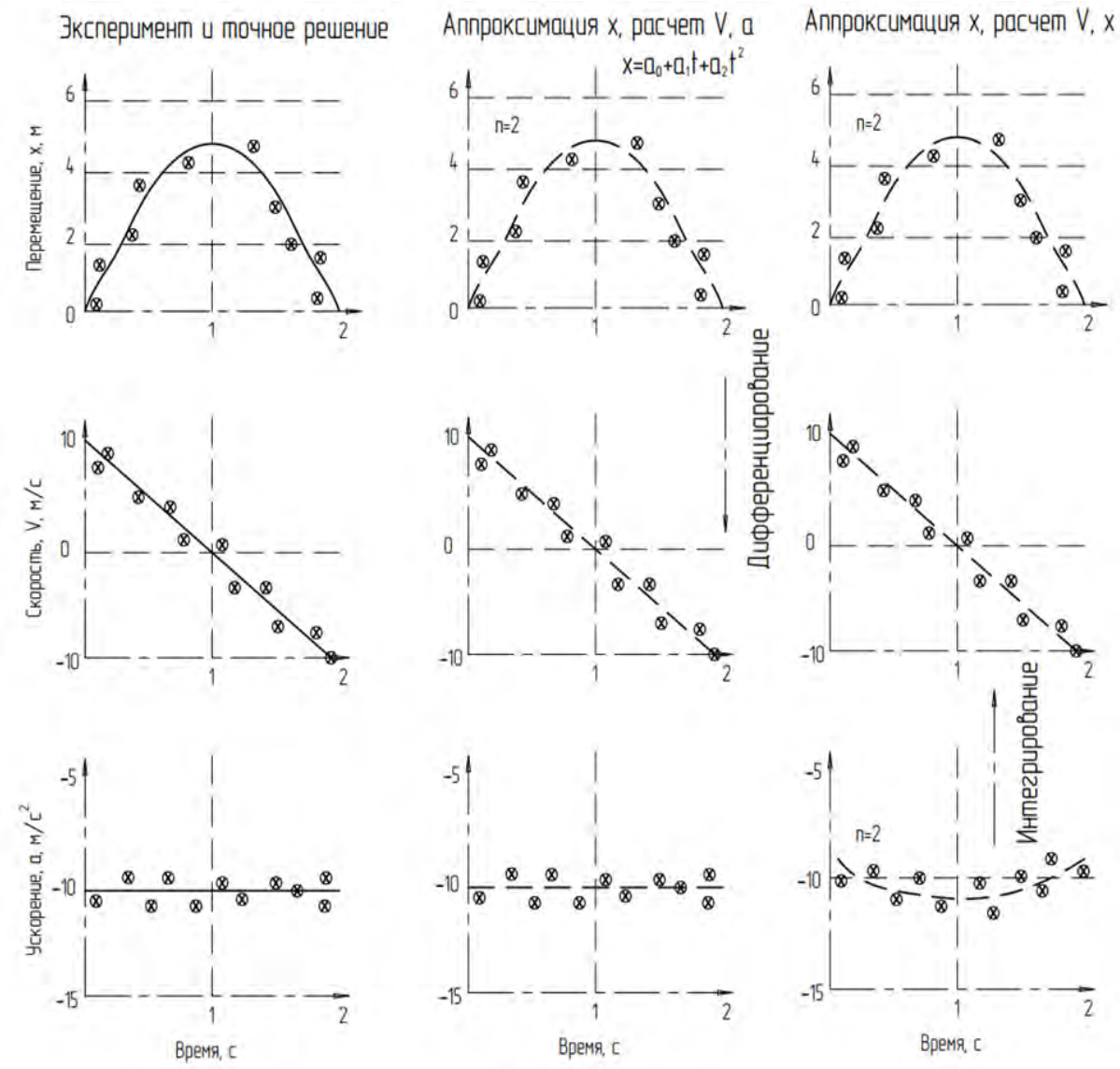


Рисунок 1.3 – Стохастический подход. Интегрирование

Для того чтобы определить недостающие величины – скорость и перемещение необходимо провести интегрирование. Производная перемещения x – это скорость, а производная скорости V – ускорение.

В данном случае, оба этих подхода эквивалентны. Аппроксимацию a можно записать как $a = Q_0+Q_1t+Q_2t^2$.

Простой подход заключается в том, что искусственный интеллект предназначен для решения задач, которые традиционно считаются прерогативой человеческого разума. Искусственный интеллект – это область информатики, которая занимается моделированием разумного поведения с помощью цифровых устройств. С другой стороны, если при решении

задачи используются методы искусственного интеллекта, такие как машинное обучение, глубокое обучение и т.д., то такая задача может считаться относящейся к искусственному интеллекту.

Структура и содержание искусственного интеллекта в общем виде представлено на рисунке 1.4. Включает в себя такие области как – машинное обучение (machine learning), обучение представлениям (representation learning) и глубокое обучение (deep learning).

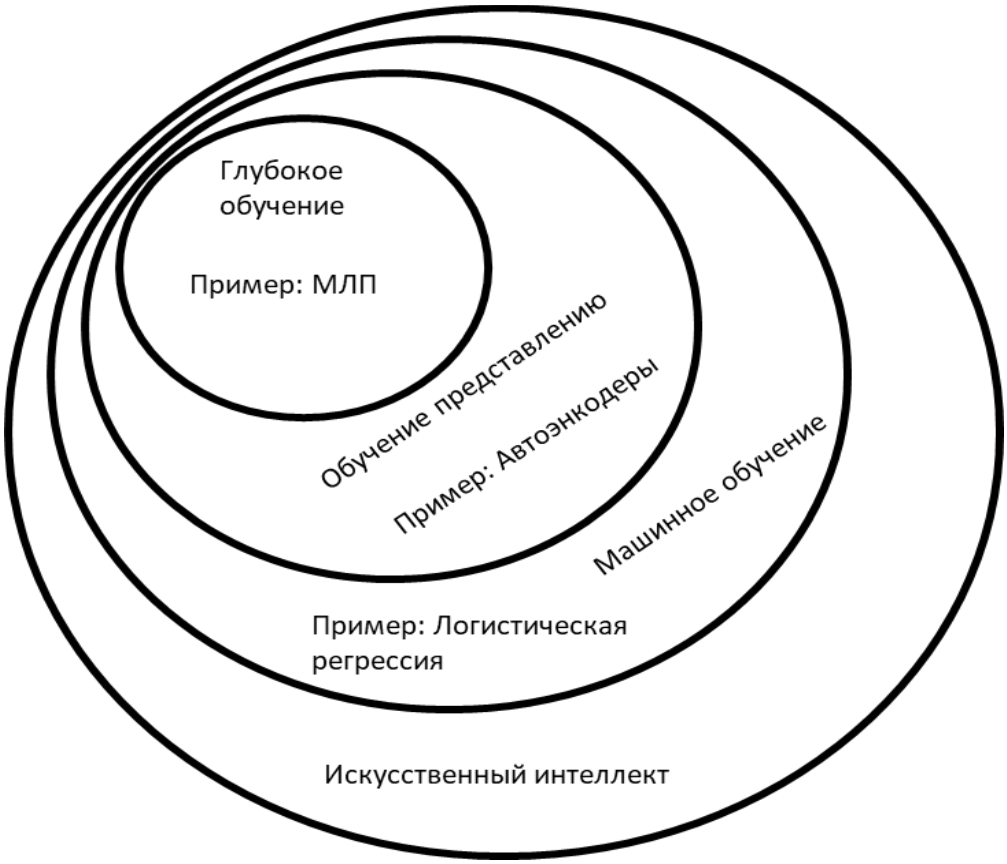


Рисунок 1.4 – Содержание ИИ

Машинное обучение – это более устоявшаяся область, которая изучает, как компьютеры могут обучаться без явного программирования. Основные атрибуты машинного обучения включают наличие задачи, наличие обучающего набора данных и необходимость обучения программы на основе этого набора данных. Или же другими словами атрибуты машинного обучения – это наличие задачи «З» в ходе решения которой программа обучается из опыта «О» и повышает меру качества «К».

В общем случае искусственный интеллект можно рассматривать на уровне идеи, которая реализуется на основе разных подходов, в т.ч. с помощью машинного обучения и для реализации этих подходов используются различные средства: *Идея (AI) + Подход (ML) + Средства (ANNs, Dataset)*, где AI – искусственный интеллект; ML – машинное обучение; ANNs искусственные нейронные сети, Dataset – базы данных.

Искусственный интеллект охватывает широкий спектр подходов, начиная от классических методов программирования, в которых вся последовательность действий явно прописывается программистом, до методов машинного обучения, в которых принятие решений происходит внутри программы без явной инструкции (рис. 1.5).

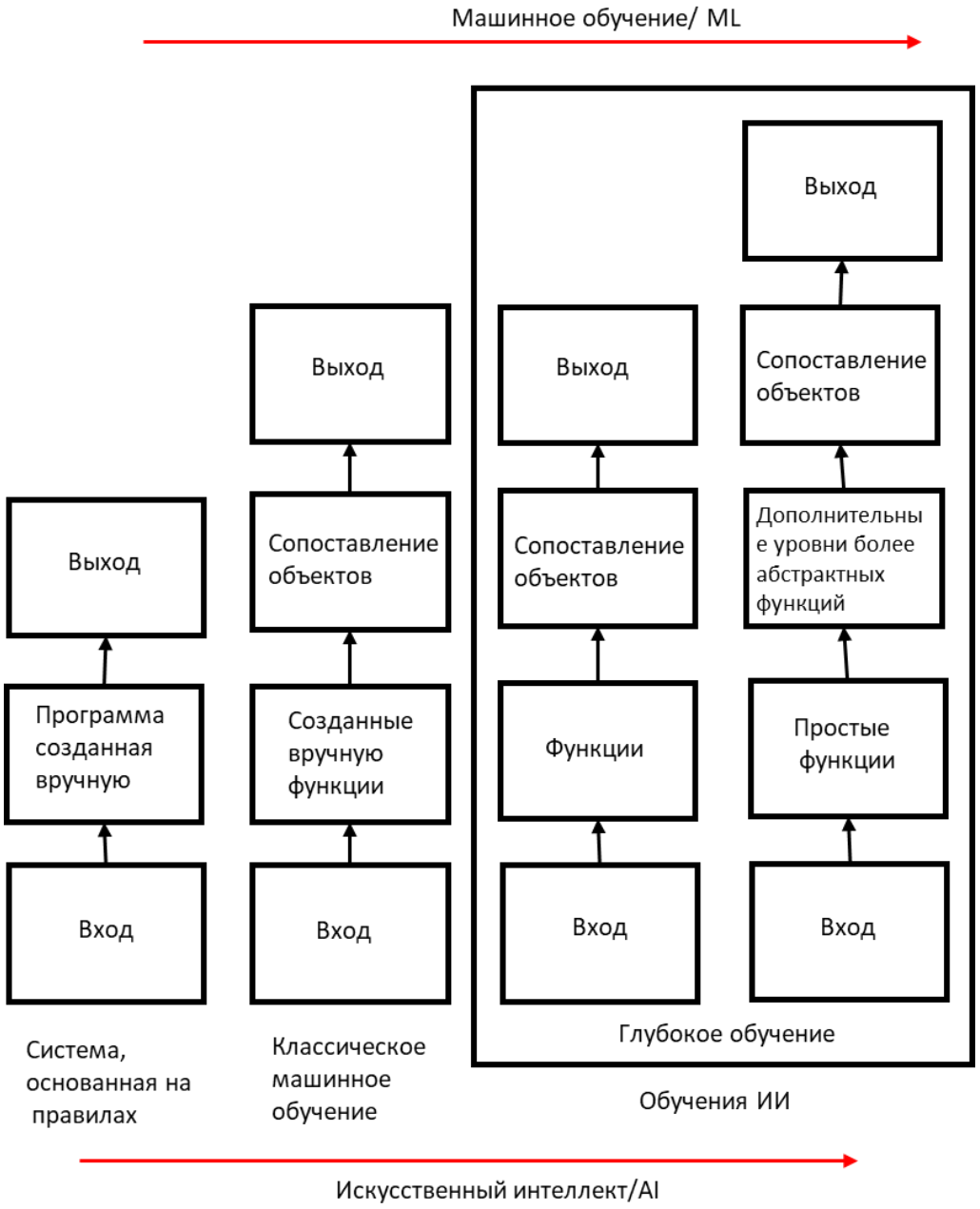


Рисунок 1.5 – Алгоритм машинного обучения в рамках ИИ

Глубокое обучение, или использование глубоких искусственных нейронных сетей, является пиком развития в области искусственного интеллекта на сегодняшний день. Основная идея заключается в том, что программы способны решать задачи, которые раньше считались прерогативой только человеческого разума.

Первый подход, представлен на рисунке 1.5 в левой колонке, заключается в определении всех признаков, которые могут помочь определить, изображен круг или квадрат на картинке. Может использоваться матрица, в которой каждый элемент представляет собой интенсивность цвета пикселя: белый цвет соответствует значению 1, а черный - 0. Можно определить границы изображения, сравнивая значения соседних пикселей. Затем производится нахождение расстояния от каждой точки до всех остальных и найти минимальное расстояние до них – это расстояние будет соответствовать центру фигуры. Подобным образом можно определить оси симметрии фигуры.

Второй подход связан с использованием машинного обучения. Данный подход требует большого количества данных, например изображения кругов и квадратов, чтобы программа могла учиться на них. Программа должна уметь оценивать качество своего решения для каждого образца. В процессе обучения программа будет улучшать свои результаты и уменьшать количество ошибок. В результате получится программа, которая сможет загружать новые изображения и определять, что на них изображено.

В классическом машинном обучении существует два основных подхода (рис. 1.6): обучение без учителя и обучение с учителем. Обучение без учителя предполагает, что программа сама учится находить закономерности в данных, без явного указания на то, какие признаки нужно искать. Обучение с учителем предполагает, что есть набор образцов, для которых известны правильные ответы, и программа обучается, пытаясь минимизировать ошибку в своих предсказаниях. Обучение с учителем обычно дает более точные результаты, поэтому оно более популярно.

Рисунок 1.6 – Направления машинного обучения

Обучение с учителем и без учителя – это два разных подхода в машинном обучении. В обучении с учителем программа обучается на наборе данных, где для каждого объекта известен его класс. Программа учится определять принадлежность новых объектов к тем или иным классам.

В обучении без учителя программа сама определяет структуру данных, не имея никаких априорных знаний о классах объектов. Она выявляет закономерности и разделяет данные на группы (кластеры), исходя из сходства объектов внутри группы и различия между группами.

Для каждого нового объекта программа автоматически определяет, к какому классу (кластеру) он относится. Обучение с подкреплением – это еще один подход в машинном обучении, который отличается от обучения с учителем и обучения без учителя. В этом подходе программа не получает прямых указаний о том, какие действия правильные, а какие нет. Вместо этого она получает награду (подкрепление) за правильные действия и наказание за неправильные. Программа учится выбирать действия, которые приводят к наибольшему подкреплению, и отбрасывать те, которые приводят к наказанию.

Для успешного освоения основ машинного обучения необходимо рассматривать три теоретические дисциплины: линейная алгебра, математический анализ и теория вероятности (рис. 1.7).

Рисунок 1.7 – Основы машинного обучения

С точки зрения линейной алгебры, из-за того, что происходит действие с матрицами больших размеров, стоит напомнить определение обозначений. Основные обозначения линейной алгебры представлены в таблице 1.1.

Таблица 1.1

Основные обозначения линейной алгебры

Название	Обозначение в N-мерном пространстве	Количество компонент
Скаляр (тензор 0-го ранга)	a	$N^0=1$
Вектор (тензор 1-го ранга)	$\vec{a}=A=[\|a_{ij}\|]$	$N^1=N$
Тензор (2-го ранга)	$A=[\|a_{ij}\|]$	N^2
Тензор (n-го ранга)	$A==[\|a_{i\ldots j}\|]$	N^n
Матрица – строка (столбец)	$A=(a_i)$	N^1
Матрица	$A=(a_{ij})$	N^2

Матрица – это «таблица чисел». Матрица состоит из чисел, но они организованы в виде двумерного массива или таблицы. Матрицы используются для представления систем координат, линейных преобразований и во многих других случаях. Вектор является частным случаем тензора, но не наоборот. Скаляры, векторы, тензоры и матрицы - все это математические объекты, и каждый из них имеет свое применение. Выбор системы координат влияет на компоненты вектора, но не на компоненты матрицы

или тензора. Количество компонентов, характеризующих скалярную величину, равно нулю. Геометрически скалярная величина может быть представлена точкой, а физически – например, массой или температурой. Вектор – это более сложный объект, который обычно обозначается стрелкой над буквой. Геометрически вектор может быть представлен стрелой, а физически – силой или ускорением. Компоненты вектора обозначаются в квадратных скобках с индексом, соответствующим их позиции в векторе. Например, вектор $a = (1, 2, 3)$ имеет три компонента: $a_0 = 1$, $a_1 = 2$, $a_2 = 3$. Тензор – это математический объект, который можно рассматривать как многомерную матрицу. Тензоры могут иметь различные ранги, и компоненты тензора обозначаются с использованием индекса для каждой размерности тензора. Например, тензор второго ранга T имеет компоненты T_{ij}, где i и j являются индексами для двух размерностей тензора. Матрица может быть представлена как двумерный массив чисел или элементов другого типа. Строка и столбец матрицы определяются их индексами, которые начинаются с нуля. Компоненты матрицы обозначаются с помощью двух индексов, соответствующих их позициям в матрице. Например, если матрица A имеет размерность m × n, то ее компоненты обозначаются как a_{ij}, где i – номер строки, а j – номер столбца. Основные операции над матрицами включают сложение, умножение, транспонирование и обращение матриц.

В математическом анализе исследуются тензорные поля и с ними могут выполнятся несколько операций: градиент, дивергенция и ротор. В машинном обучении чаще всего пользуются операцией градиента (формула 1.4).

$$\nabla_a = \left[\left[\frac{\partial a}{\partial x_i}\right]\right]. \tag{1.4}$$

Матрица второго порядка имеет размерность 2x2 и состоит из четырех компонентов a, b, c, d. Для нахождения градиента функции используют частные производные по каждой координате. Градиент функции показывает направление наибольшего изменения функции и используется для оптимизации функций.

Теория вероятностей является важным инструментом в машинном обучении, так как позволяет анализировать данные и делать предсказания на основе вероятностных моделей. Основные обозначения теории вероятности рассчитываются по формуле 1.5:

$$E(X) = \mu = \sum_{i=1}^{m} x_i p(x)_i \,. \tag{1.5}$$

Важное центральное значение имеет понятие от математического ожидания. В основу функций качества положено условия максимума правдоподобия.

1.2. Линейная и полиномиальная регрессия

Регрессионный анализ используется для оценивания взаимосвязи между зависимой переменной (переменной, представляющей результат или цель) и одной или несколькими независимыми переменными (переменными, представляющими факторы или причины). Основная идея линейной регрессии заключается в поиске наилучшей линейной функции зависимости множества $Y=((y_i))$ от множества $X=((x_i))$ $(i=1...m)$ по критерию минимума некоторой функции качества для описания взаимосвязи между зависимой и независимыми переменными. Функция качества используется для оценки качества модели регрессии и определения наилучших параметров модели.

Несмотря на то, что эта задача может показаться очень простой, но она лежит в основе всего машинного обучения и в искусственных нейронных сетях. Пример: при определении линейной зависимости $y(\theta_0, \theta_1) = \theta_0 + \theta_1 x$ (рис. 1.8) между набором входных данных значений x какого-то количества m штук значений и набор значений выходных данных y. Задаются входные данные и неизвестные θ (необходимо найти θ_0 и θ_1) в виде матриц: $X = \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(m)} \end{pmatrix}$; $Y = \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix}$; $\Theta = \begin{pmatrix} \theta^{(1)} \\ \dots \\ \theta^{(m)} \end{pmatrix}$.

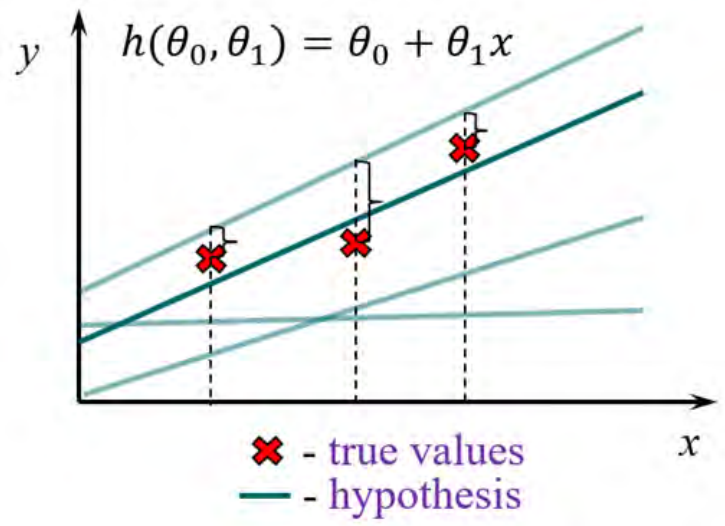


Рисунок 1.8 – Решение задачи с помощью машинного обучения

Методы решения обратных задач могут различаться в зависимости от конкретной задачи и имеющихся данных. Один из распространенных методов – это метод наименьших квадратов, который заключается в минимизации суммы квадратов ошибок между наблюдаемыми данными и моделью. Также могут использоваться методы регуляризации, чтобы избежать переобучения модели. В общем случае, решение обратной задачи требует применения специализированных математических методов и алгоритмов.

Нужно это для того, чтобы, задавая какое-то конкретное их значение, можно было определить значение гипотезы. Гипотеза h должна быть приближена к экспериментальным данным y. В процессе решения задачи нужно определить наилучшие значения. Для решения подобной задачи необходимо посчитать суммарную квадратичную ошибку для сокращение отрицательных положительных ошибок. В таком случае функция которая минимизирует, выглядит следующим образом:

$$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(h^{(i)}-y^{(i)})^2 \Rightarrow \min \qquad (1.6)$$

В скобках указана ошибка в виде разности между значением выдаваемой гипотезы и значением реальным, разность возводится в квадрат и производится суммирование по всем данным.

В более сложном случае объём данных гораздо больше. В таком случае функцию J можно представить в пространстве аргументов θ_0 и θ_1 (рис. 1.9). Значение функции качества J зависит аргументов θ_0 и θ_1.

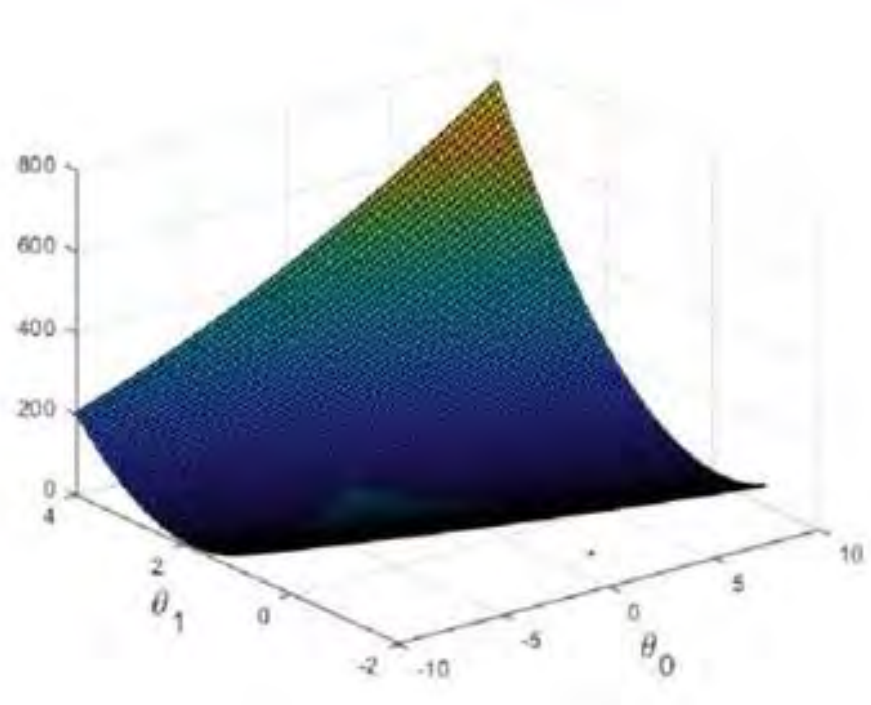


Рисунок 1.9 – Множество значений x первый вид графика

Таким образом паре значений θ_0 и θ_1 будет соответствовать определённое значению функции качества. При определении значений θ_0 и θ_1 при которых функция качества будет принимать минимальное значение, можно представить в виде контурного графика (рис. 1.10). На графике каждая линия представляет собой постоянное значение искомой функции J.

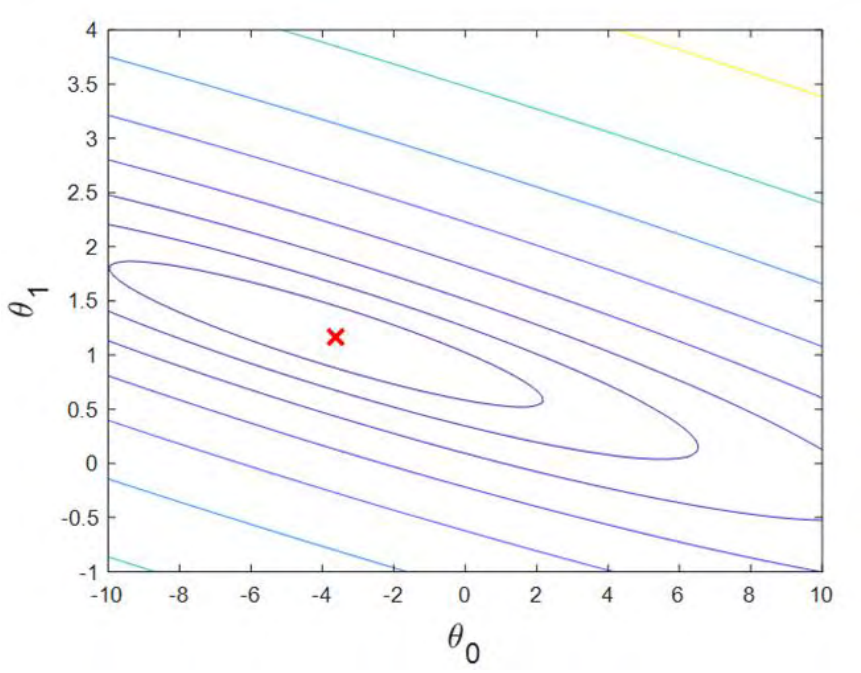


Рисунок 1.10 – Множество значении θ_0 и θ_1 в виде контурного графика

Метод градиентного спуска используется для минимизации функции качества $J(\theta)$, где θ – вектор параметров модели. Суть метода заключается в том, что на каждой итерации происходит движение в направлении антиградиента функции, то есть в направлении максимального убывания функции. Это позволяет найти локальный минимум функции качества и улучшить качество модели.

При машинном обучении распространен метод градиентного поиска (спуска/подъёма). Метод градиентного спуска заключается в следующем:

1. берется начальная точка θ_0 и вычисляется значение функции качества $J(\theta_0)$;
2. вычисляются частные производные функции качества по каждому параметру в точке θ_0. Данные производные образуют вектор градиента, который указывает направление наибольшего увеличения функции;
3. далее делается шаг в направлении, противоположном градиенту, на некоторую величину α. Величина α называется размером шага и выбирается экспериментально;
4. затем повторяется процесс вычисления градиента и шага в его направлении для новой точки. Этот процесс продолжается до тех пор, пока функция качества не достигнет минимума или не будет выполнено определенное количество итераций.

Действия с матрицами. Матрица A имеет строки и столбцы. Номера строк указывают первым индексом i, номера столбцов вторым j. Если проводить аналогию с тензорным анализом, то матрица 1.7 будет относится к матрицам второго ранга (так как имеет два индекса i и j) записанной в трехмерном пространстве (имеет три строки и три столбца).

$$A = ((a_{ij})) = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}. \quad (1.7)$$

Правило Эйнштейна: если в одночлене содержащем индексы или одинаковые с индексом переменные встречаются повторяющиеся индексы или переменные, то по этим индексам необходимо провести суммирование (пример: $a_{ij}=a_{11}+a_{22}+a_{33}$ и $ia_i=1a_1+2a_2+3\ a_3$).

Исключение из правила Эйнштейна – исключение Лурье: если повторяющиеся индексы встречаются с двух сторон от знака равенства, неравенства или тождества, то по этому индексу суммирование производить нельзя (пример: $c_i=a_{ij}b_i$).

Действия с базовыми матрицами:

1. Умножение матрицы на число: $C = \lambda A$, $c_{ij} = \lambda a_{ij}$;
2. Сложение матриц: $C=A+B$, $c_{ij} = a_{ij} + b_{ij}$;
3. Транспонирование матрицы: A^T, $a_{ij}^T = a_{ji}$;
4. Обратная матрица: A^{-T}, $AA^{-1}=I$, $I =[|\delta_{ij}|]\$$
5. Умножение матриц: $C = AB$, $c_{ij} = a_{ik} \cdot b_{kj}$;
6. Дифференцирование матриц: $\frac{\partial A}{\partial x} = \left(\left(\frac{\partial a_{ij}}{\partial x}\right)\right)$;
7. Интегрирование матриц: $\int A dx = \left(\left(\int a_{ij} dx\right)\right)$.

Для определения значения гипотезы для каждой пары экспериментальных данных от *1* до *m,* формулу 1.6 можно записать в матричной форме, т.е. определить матрицу по всем экспериментальным точкам.

Необходимо матрицу $X = \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(m)} \end{pmatrix}$ дополнить столбцом единиц и получаем матрица вида $H = \begin{pmatrix} 1 & x^{(1)} \\ \dots & \dots \\ 1 & x^{(m)} \end{pmatrix}$. Индексы указаны в скобках для того, чтобы не возникло ошибки восприятия со степенями. Далее $H = \begin{pmatrix} 1 & x^{(1)} \\ \dots & \dots \\ 1 & x^{(m)} \end{pmatrix} \cdot \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$, при этом размерность матрицы x равна [m; 2] m-строк и 2 столбца, размерность матрицы θ две строки и один столбец [2; 1]. В процессе произведения размерность сократится и получится [m; 1]. Например первое произведение примет вид $h^{(1)} =1\ \theta_0+x^{(1)}\ \theta_1$ и т.д. Преимущества подобной записи при программной реализации заключается в том, что такой расчёт (умножение двух матриц) выполняется быстрее чем цикл по расчёту m раз величины h. Если значение h, то это значение можно подставить в выражение для поиска минимума ошибки, определить ошибку для случайного состояния θ.

Алгоритм для применения метода градиентного спуска заключается в следующем: необходимо определить компоненты градиента $\nabla J = \left[\left|\frac{\partial J}{\partial \theta_0}\frac{\partial J}{\partial \theta_1}\right|\right]$. Первая производная по θ_0 равна, $\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h^{(i)} - y^{(i)})x^{(i,1)}$, вторая производная по θ_1 равна $\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}(h^{(i)} - y^{(i)})x^{(i,2)}$.

При известном компоненте градиента ∇J, то при поиске минимума необходимо выполнить рассмотрение вектора в обратную сторону $\vec{g} = -\alpha\nabla J$. Знак минуса указывает на обратное направление относительно вектора ∇J, а коэффициент α – это скорость поиска минимума.

Алгоритм поиска минимума функции качества:

1. Задать начальные значения компонентов матрицы θ_0 и θ_1 случайным образом (например генератором случайных чисел).

2. Рассчитываются значения искомой функции J и компоненты градиента ∇J.

3. Определяются новые значения θ_0 и θ_1:

$$\theta_0^{н} = \theta_0^{с} - \alpha\frac{\partial J}{\partial \theta_0} \tag{1.8}$$

$$\theta_1^{н} = \theta_1^{с} - \alpha\frac{\partial J}{\partial \theta_1} \tag{1.9}$$

где $\theta_i^{н}$ – новое значение компонентов матрицы;

$\theta_i^{с}$ – старое значение компонентов матрицы.

В выражениях 1.8 и 1.9 представлено изменение данных значений θ_0 и θ_1 смещением на компоненту градиента взятую со знаком минус и умноженное на коэффициент α.

4. Переход в новую точку пространства осуществляется следующим образом - повторять пункты 2-3 до достижения выполнения одного из условий $J^{н} - J^{с} \le \delta$ или $N_{итераций} \ge N_{max}$, где $J^{н}$ – новое значение функции; $J^{с}$ – старое значение функции; δ – заданная ошибка; $N_{итераций}$ – количество итераций; N_{max} – максимальное количество шагов.

5. Вывод результатов: θ_0, θ_1 и h.

Резюмировать изложенные пункты можно следующим образом: Производится случайный выбор начальной точки θ_0, θ_1, в ней определяется направление градиента и делается шаг в обратном направлении. Далее повторяются эти действия до нахождения искомого минимума. При этом решение будет содержать некую ошибку между численным и реальным значением.

Батч (Batch) – на каждом шаге градиентного спуска используются все обучающие данные (все m-штук элементов). В современных алгоритмах используются мини-батчи, т.е. часть этих данных на каждом шаге.

На рисунке 1.11 представлена классификация видов регрессии.

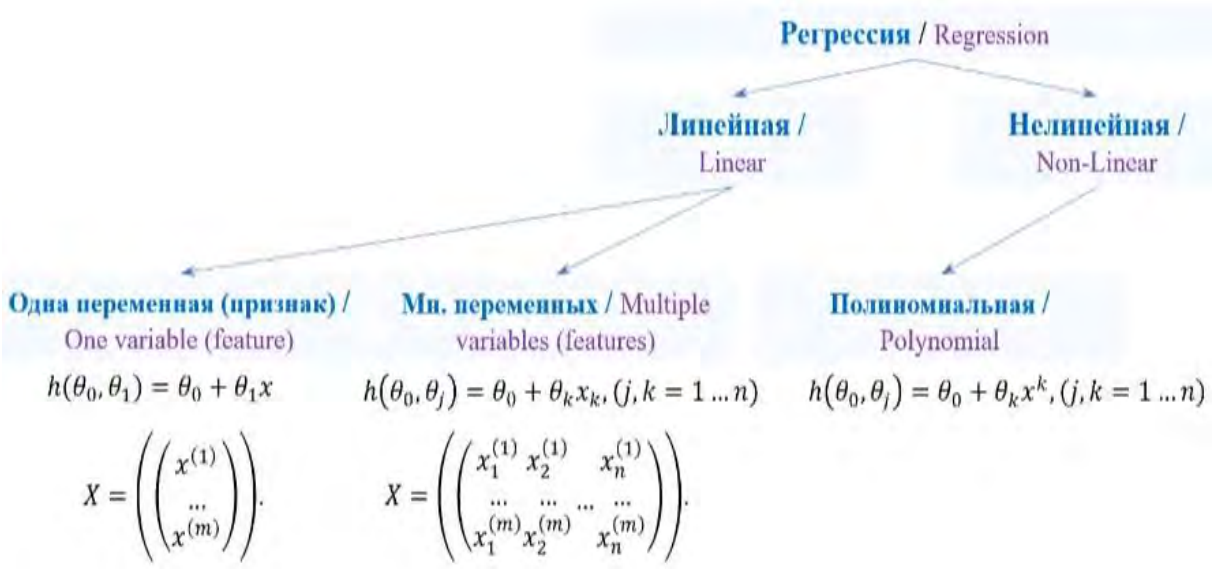


Рисунок 1.11 – Виды регрессии

Таким образом рассмотрена в регрессионном анализе линейная функция одной переменной или с точки зрения машинного обучения – признак. Имеется исходные данные *X*, *Y*, эти данные используются для обучения, например, методом градиентного спуска. Определяются параметры рассматриваемой модели θ_0, θ_1 и находится значения модели *h*. Используя новые входные данные на основе этой модели можно получать оценку, прогноз согласно новым данным.

В случае многих переменных, когда используется несколько признаков, следует использовать правило Эйнштейна. В таких случаях матрица входных *X* на основании экспериментов от 1 до *m* будет иметь вид: $\begin{matrix} X_1 \\ X_2 \\ \dots \\ X_k \end{matrix} \rightarrow Y(X_k)$ исходя из данного выражения матрица примет вид $X = \begin{pmatrix} 1 & x_1^{(1)} & \dots & x_n^{(1)} \\ \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & \dots & x_n^{(m)} \end{pmatrix}$. Полученная матрица включает все входные переменные и дополнительный столбец с единицами. Матрица *Y* имеет вид $Y = \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix}$; матрица θ имеет компоненты θ_0, θ_1 и до θ_n: $\Theta = \begin{pmatrix} \theta_0 \\ \dots \\ \theta_n \end{pmatrix}$. При этом можно сразу

определить все значения H путём составления объединённой матрицы $H=X\,\theta$. Для реализации данного алгоритма необходимо определить компоненты градиента по выражению: $\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h^{(i)} - y^{(i)})x^{(i,j)}$.

В случае недостаточности линейной регрессии применяют полиномиальную регрессию.

Полиномиальная регрессия – это метод регрессионного анализа, который используется для описания более сложных зависимостей между переменными. Он основан на использовании полиномиальных функций для аппроксимации данных. Это может быть полезно, когда зависимость между переменными не может быть описана линейной функцией.

Если зависимость нелинейная, то для такой аппроксимации необходимо использовать полином. В развернутом виде модель выражается как: $h = \theta_0 + \theta_1 x + \theta_1 x^2 + \theta_1 x^3 + \ldots$. Необходимо ввести новое обозначение. Вместо x^2 вводится x_2, вместо x^3 вводится x_2 и т.д. $x^i \rightarrow x_i$. В этом случает, решение задачи сводится определению функции многих переменных.

1.3. Логистическая регрессия

Логистическая регрессия имеет важное значение для машинного обучения, так как все задачи распознавания основаны на ней.

Логистическая регрессия – это метод анализа данных, который использует математику для поиска взаимосвязей между двумя факторами данных. Затем эта взаимосвязь используется для прогнозирования значения одного из этих факторов на основе другого. Предсказание обычно имеет конечное количество результатов, например, «да» или «нет».

Общая классификация регрессии представлена на рисунке 1.12.

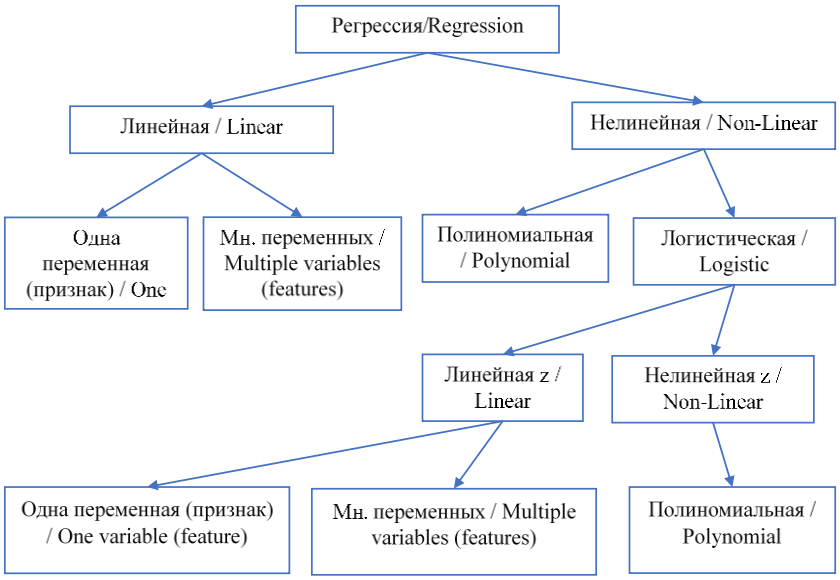


Рисунок 1.12 – Классификация регрессии

Логистическая регрессия является отдельным классом задач регрессии. Основная идея заключается в поиске наилучшей функциональной зависимости бинарного множества $Y = ((y_i))$ от множества $X = ((x_i))$ при $(i = 1...m)$ по критерию минимума некоторой функции качества. То есть Y – это бинарное множество для некоторого набора значений, которое может принимать значение либо ноль, либо единица. В таком случае использование прямой линейной регрессии даст очень большую ошибку, то есть возникнут большие отклонения от наблюдаемых данных.

Исходя из этого возникла идея логистической регрессии, которая реализуется в два шага. Шаг первый – это произвести замену переменных, то есть X поменять на Z, таким образом, в самом простом случае Z линейно зависит от X. В результате замены переменных обозначает, ось Z смещается относительно X, это нужно для того, чтобы ось Z разделила данные на два класса (рисунок 1.13).

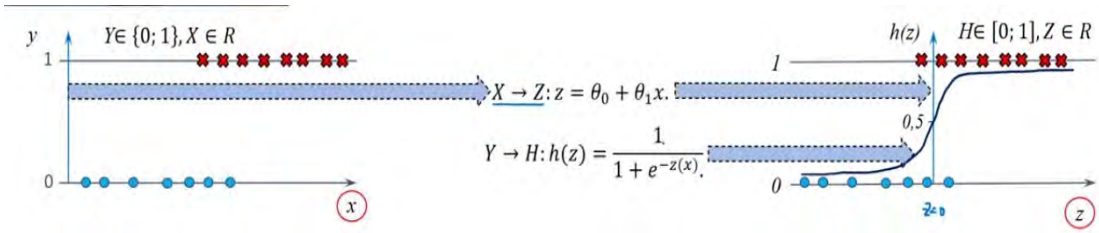


Рисунок 1.13 – Замена переменных

В ходе выполнения второго шага – для переменных Z определяется гипотеза в виде, представленном на рисунке 1.13. Это так называемая логистическая функция, то есть единица, деленная на один плюс e в степени минус Z. Остается подобрать параметры полученной функции, то есть определить положение Z (сжимать или растягивать), чтобы наилучшим образом описать данные.

Тогда задаётся правило: если гипотеза, или ответ, программы будет меньше 0,5, то программа дает ответ ноль. Если больше или равно 0,5, то ответ – единица. Таким образом работает машинное обучение.

Например, если нужно определить, болен или здоров пациент и есть какие-то данные, то программа выдает гипотезу и по правилам округления выдается результат – ноль или единица. Решается задача похожим образом. Исходные данные – X и Y, X дополняются столбцом из единиц, далее матрица θ неизвестная, матрица Z задается как произведение X на θ, аналогичные действия выполнялись в линейной регрессии, но далее другой этап – гипотеза считается такой нелинейной функцией и этот этап невозможно представить в матричном виде. Но типичную из прошлой задачи линейной регрессии функцию качества минимизировать не удобно, так как она имеет вид с множеством локальных экстремумов. Поэтому используется более сложная на вид функция, логарифмическая, которую нужно минимизировать. Здесь можно

убедиться, что совпадение *Y* и *H* дает малое значение данной функции – *Y* имеет значение либо ноль, либо единица (рис. 1.13). При близких значения *Y* и *H* между собой функция является достаточно малой.

Это и есть принцип максимального правдоподобия. Опять же, здесь он выводит точно такую же функцию. Для программной реализации нахождение такой функции не является сложной задачей (рис. 1.14). То есть в задачах логистической регрессии очень часто берут подобную функции, либо используют только ее половину.

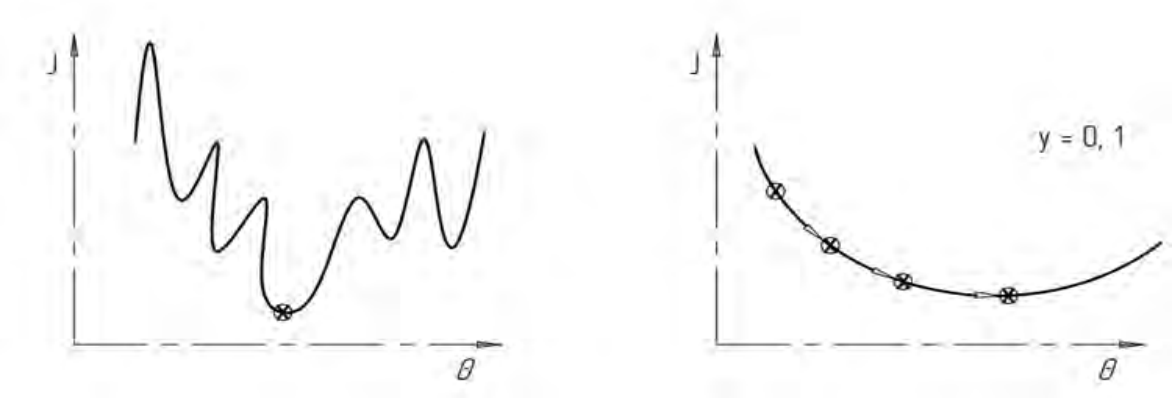


Рисунок 1.14 – Минимизация функции

Если входной фактор один, то процедура решения следующая. При известных данных *X*, к значениям *X* добавляется единица, определяется новая переменная *Z*. Затем устанавливается гипотеза. Зная гипотезу можно посчитать ошибку. И градиент функции оказывается точно таким же, как функция линейной регрессии (формулы 1.10-1.14):

$$X = \begin{pmatrix} \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(m)} \end{pmatrix} \end{pmatrix}; Y = \begin{pmatrix} \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix} \end{pmatrix} \tag{1.10}$$

$$X = \begin{pmatrix} \begin{pmatrix} 1 & x^{(1)} \\ \dots & \dots \\ 1 & x^{(m)} \end{pmatrix} \end{pmatrix}; \theta = \begin{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} \end{pmatrix}; Z = X \cdot \theta \tag{1.11}$$

$$h(z) = \frac{1}{1 + e^{-z(x)}} \tag{1.12}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \ln(h^{(i)}) + (1 - y^{(i)})(\ln(1 - h^{(i)})) \Rightarrow \min \tag{1.13}$$

$$\nabla J = \frac{1}{m} X^t (H - Y). \tag{1.14}$$

Алгоритм поиска минимума функции качества в данном случае выглядит следующим образом:

1. задать начальные значения Θ;
2. рассчитать значения H ($H=X\cdot\Theta$) и ∇J по формуле 1.14;
3. найти $\Theta^{н}=\Theta^{с}-\alpha\nabla J$;
4. повторять пункты 2-3 до выполнения одного из условий: $J^{н}-J^{с}<\delta$, коэффициент итерации $> N_{max}$;
5. вывод результатов: Θ.

Это и является решением задачи.

Если входных факторов два, то есть регрессия относится к виду с множеством переменных (признаков), при этом процедура решения следующая: матрица X имеет два столбца. Матрица Y – нули, единицы, диагнозы и прочее. Далее матрица X дополняется единицами, умножается на матрицу неизвестных коэффициентов, затем стандартная процедура – Z это произведение этих матриц (формула 1.15-1.17).

$$X=\begin{pmatrix}\begin{pmatrix}x_1^{(1)} & x_2^{(1)}\\ \dots & \\ x_1^{(m)} & x_2^{(m)}\end{pmatrix}\end{pmatrix};Y=\begin{pmatrix}\begin{pmatrix}y^{(1)}\\ \dots\\ y^{(m)}\end{pmatrix}\end{pmatrix}. \tag{1.15}$$

$$X=\begin{pmatrix}\begin{pmatrix}1 & x_1^{(1)} & x_2^{(1)}\\ \dots & \dots & \dots\\ 1 & x_1^{(m)} & x_2^{(m)}\end{pmatrix}\end{pmatrix};\theta=\begin{pmatrix}\begin{pmatrix}\theta_0\\ \theta_1\\ \theta_2\end{pmatrix}\end{pmatrix};Z=X\cdot\theta. \tag{1.16}$$

$$h(z)=\frac{1}{1+e^{-z}}. \tag{1.17}$$

Затем определяется гипотеза – функция, которую нужно минимизировать. Благодаря тому, что часть действий происходит в матричной форме, алгоритмы универсальны, поэтому $J(\theta)$ определяется по формуле 1.13. Все это повторяется и находится значение θ.

Рассмотрим более общий случай, когда количество входных факторов больше двух. Представим, что нужно решить задачу распознавания изображения. Например, нужно определить, что изображено – девятка ли на этом изображении? Если да, то ответ сети единица, если нет, то ответ – ноль. В данном случае параметров очень много (рис. 1.15).

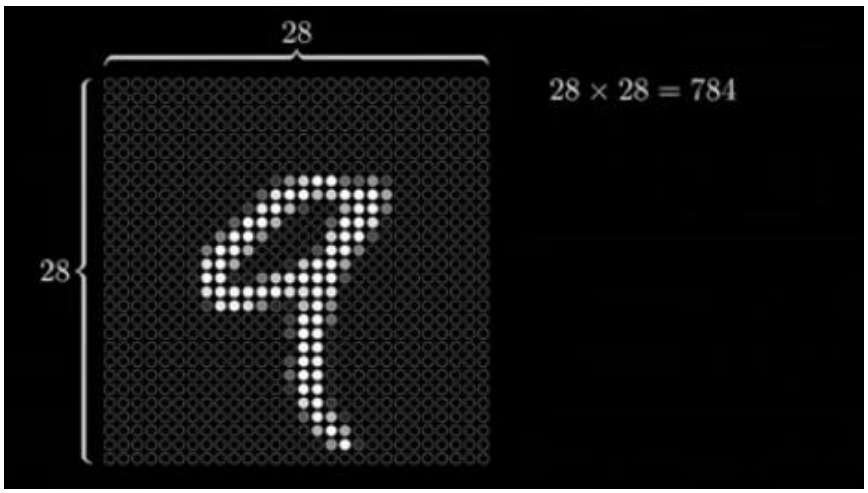


Рисунок 1.15 – Исходное изображение

Для представленного изображения количество X равно количеству пикселей. То есть получается одна картинка – это огромная матрица в семьсот восемьдесят четыре элемента. Далее применяется стандартная процедура. Итоговая матрица Z будет размером $[m, 1]$, то есть сколько картинок, столько и будет этих матриц Z. Дальше для всех значений всех элементов матрицы определяется гипотеза – определяется функция, которую надо минимизировать. Находятся новые значения, и повторяются вышеперечисленные шаги и в итоге определяется матрица θ (формулы 1.18-1.21).

$$X = \begin{pmatrix} \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ \dots & & \dots & \dots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \end{pmatrix}; Y = \begin{pmatrix} \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix} \end{pmatrix}. \tag{1.18}$$

$$X = \begin{pmatrix} \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \end{pmatrix}; \theta = \begin{pmatrix} \begin{pmatrix} \theta_0 \\ \dots \\ \theta_n \end{pmatrix} \end{pmatrix}; Z = X \cdot \theta. \tag{1.19}$$

$$h(z) = \frac{1}{1 + e^{-z(x)}}. \tag{1.20}$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \ln(h^{(i)}) + (1 - y^{(i)})(\ln(1 - h^{(i)})) \Rightarrow \min \tag{1.21}$$

Матрица θ подразумевает, что готовая программа может определить, девятка на картинке или нет. Если нет, то она выдает ноль или близкое к нулю значения, а если да, то близкое к единице значение.

Что, если нужно аппроксимировать полином? Здесь применяется тот же самый алгоритм. Есть набор значений x_1 и x_2, есть набор ответов Y.

Но матрица *X* представляется уже в другой форме – производится замена переменных для составления матрицы (формула 1.24). В этом случае $x_3=x_1x_2$; $x_4 = x_1^2, x_5 = x_2^2$. Данная матрица умножается на матрицу θ, и в результате получаются значения *Z*. Затем находится гипотеза, функция, ее градиент. Далее функция минимизируется по стандартной процедуре. После того, как задача решена, значения коэффициентов известны, программа способна разделить, к какому классу относится значение. Если приравнять к нулю функцию (формула 1.22), то можно определить границу, разделяющую два класса – единица и ноль.

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 . \qquad (1.22)$$

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ \dots & \\ x_1^{(m)} & x_2^{(m)} \end{pmatrix}; Y = \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix}. \qquad (1.23)$$

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & x_5^{(1)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & x_4^{(m)} & x_5^{(m)} \end{pmatrix}; \theta = \begin{pmatrix} \theta_0 \\ \dots \\ \theta_5 \end{pmatrix}; Z = X \cdot \theta . \qquad (1.24)$$

Значения параметров *h(z)* и *J(θ)* определяются аналогично по формулам 1.20, 1.21.

Эту идею, или этот алгоритм, можно применять для любого количества классов, но в искусственных нейронных сетях это делается еще проще, так как не надо вручную разбивать на какое-то количество задач.

Граница принятия решений составляет 0,5 для любого из представленных примеров. То есть, если программа выдает значение, меньшее 0,5, то это класс ноль. В противном случае – класс единица.

Идею можно развить на многоклассовую классификацию – один против всех (Multi-class classification: one-vs-all). Основной смысл – поиск наилучшей функциональной зависимости мультиарного (*p*–арного) множества $Y=((y_i))$, $Y \in (1, 2, 3 \dots p)$, от множества $X=((x_i))$, $X \in R$, $(i= 1 \dots m)$ по критерию минимума некоторой функции качества *J(θ)* путем последовательного решения «*p*» задач двухклассовой классификации ($p > 2$).

1.4. Настройка машинного обучения

Машинное обучение характеризуется гиперпараметрами, то есть какие- то параметры, которые задаются пользователем и не единственным образом определяют решение и очень существенно могут влиять на результат. Например, скорость обучения – как задать, не совсем понятно, но понятно, что от ее значения зависит результат.

Масштабирование признаков в части настройки машинного обучения Диапазон значения входных параметров x_1, x_2 и так далее может существенно различаться. Например $10^{-10} \leq x_1 \leq 10^0$; $10^2 \leq x_2 \leq 10^7$, то в таких случаях при попытке определения гипотезы $H = \theta_0 + \theta_1 x_1 + \theta_1 x_2$. В ситуации, когда между значениями параметров x будет существенная разница, то при подборе параметров θ произойдёт искажение области поиска решения, что приведёт к росту ошибки.

Самый простой способ отмасштабировать признаки. Каждый признак умножают на величину масштаба: $\dot{X}_j = S_j \cdot X_j$. В таком случае значения признаков будет одинакового порядка.

Есть другие варианты масштабирования: масштабирование от нуля до единицы. $\dot{X}_j = (X_j - min(X_j))/(max(X_j) - min(X_j))$. Этот метод позволяет сделать все данные заметными, и очень часто помогает в задачах машинного обучения.

Скорость обучения α. Резкое снижении функции ошибки по номерам итерации указывает на нормальную скорость обучения. При низкой скорости обучения значение ошибки убывает незначительно. При высокой скорости обучения компоненты градиента будут иметь большие значения и может происходить перескакивание минимума. Поэтому подбор скорости обучения является важным параметром (рис. 1.16).

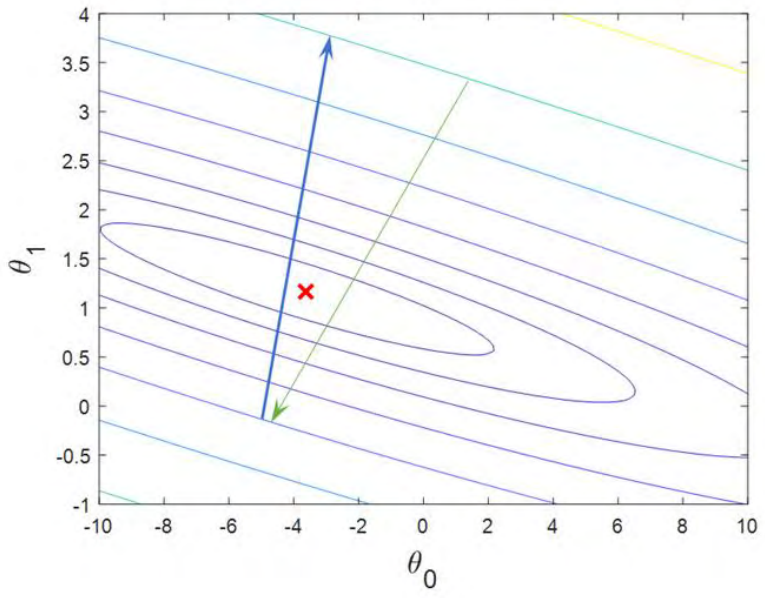


Рисунок 1.16 – Зависимость компонентов от скорости обучения

Погрешность δ и количество итераций N_{max}. Данные параметры задаются исследователем. Для определения завершения поиска необходимо сравнить значения новой и старой функции. Если разница между значениями будет меньше заранее установленной погрешности δ, то указывает на завершения процесса решения задачи.

Регуляризация. В случае если регрессия полиномиальная, то часто используется регуляризация. При n=1 – линейная аппроксимация, то часто этого недостаточно и выражается явлением – недообучение в машинном обучении. Для устранения недообучения повышают значение n, при этом можно достичь ошибки равной нулю. Но при расчёте новых данных вне имеющихся значений ошибка будет значительной и такое явление называется переобучение (рис. 1.17).

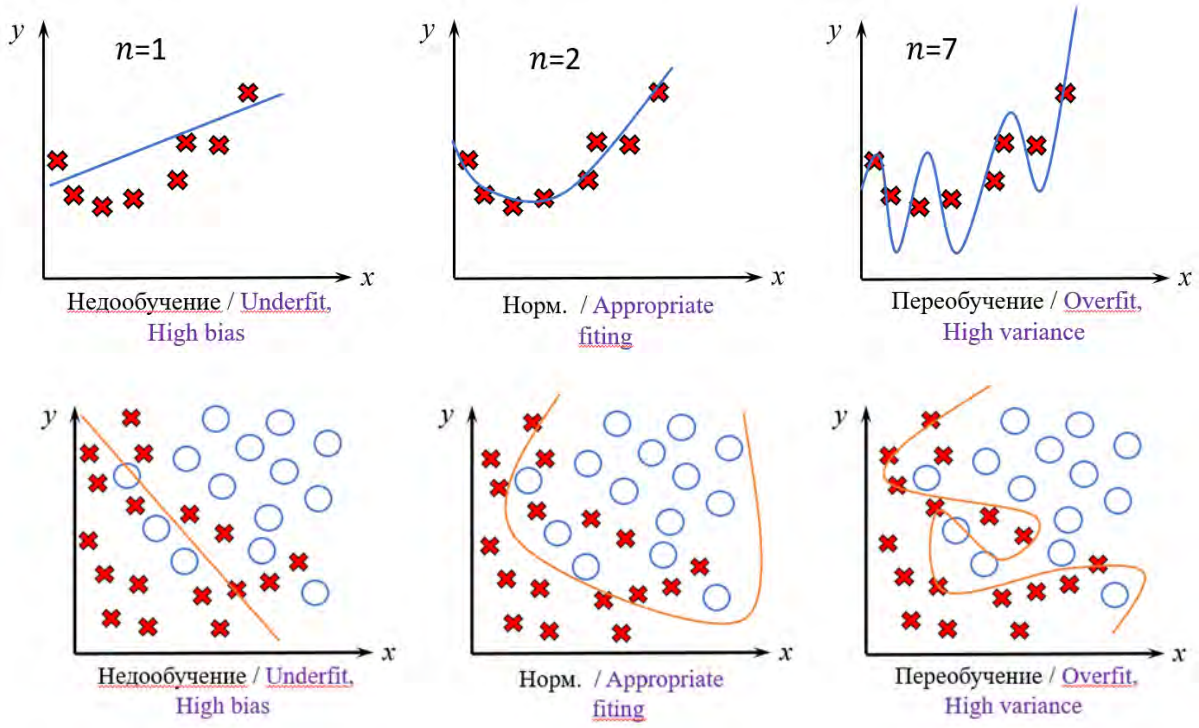


Рисунок 1.17 – Как влияет недообучение и переобучение на параметры

Весь Dataset в машинном обучении делится на три части. Первая часть – обучение (около 70 %), валидация (15 %) входе которой определяется гиперпараметр и тестирование (15 %). Сложная, переобученная модель может иметь очень много признаков (особенно если речь идёт о полиноме).

Процедура устранения переобучения сводится к тому, что функции качества дописывается слагаемое, которое «наказывает» за не нулевые значения весов параметров θ (формула 1.25).

$$J(\theta_0,\theta_j)=\frac{1}{2m}\left[\sum_{i-1}^{m}(h^{(i)}-y^{(i)})^2+\lambda\sum_{l=1}^{n}\theta_j^2\right]\Rightarrow \min \qquad (1.25)$$

То есть, если аппроксимация происходит с помощью полиномиальной функции, то добавочная функция наказывает все значения за наличие коэффициентов, кроме θ_0.

Крайне важно выбрать правильную степень полинома:

$$h(\theta_0, \theta_j) = \theta_0 + \theta_k x^k, (j, k = 1...n). \quad (1.26)$$

Если степень полинома равна единице, то это недообучение, оно будет характеризоваться тем, что и на обучающей выборке, и на наборе каких-то данных программа будет выдавать большую ошибку. В случае степени полинома, равной двум, то это будет нормальный уровень обучения – программа хорошо описывает данные. В случае переобучения полином очень большой степени, данных немного. Он опишет данные максимально точно, но для новых данных модель даст нереальное значение. То есть из-за высокой степени на новых данных программа сработает недостаточно точно.

Добавочная функция в формуле 1.25 ограничивает переобучение. Регуляризацию следует применять и при решении задач классификации. В таком случае количество компонентов, степень полинома влияет на форму границы (рисунок 1.17). Функция качества для задачи классификации будет выглядеть следующим образом:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} \ln(h^{(i)}) + (1 - y^{(i)})(\ln(1 - h^{(i)})) + \frac{\lambda}{2m}\sum \theta_j^2 \Rightarrow \min \quad (1.27)$$

Регуляризация в машинном обучении – это применение бритвы Оккама, которая гласит: «Сталкиваясь с двумя одинаково хорошими гипотезами, всегда выбирайте более простую».

Очевидно, в большинстве случае визуальный подбор степени полинома (как мы это делали ранее для линейной и логистической полиномиальных регрессий) не представляется возможным. Более того, модели с таким количеством признаков в силу их высокой сложности (complexity) склонны к переобучению.

Бороться с этим можно тремя способами:

Первый вариант. Увеличить размер обучающей выборки. Маленькая выборка снижает обобщающую способность модели, а значит повышает разброс.

Второй вариант. Уменьшить количество признаков (вручную или через алгоритм, кроме того вводить наказание за новые признаки, как например, скорректированный коэффициент детерминации, adjusted R-square). Опасность здесь – удалить нужные признаки.

Третий вариант. Использовать регуляризацию, которая позволяет снижать параметр (вес, коэффициент) признака и, таким образом, снижать его значимость.

2. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ И ОБУЧЕНИЕ С УЧИТЕЛЕМ

2.1. Математическая модель нейрона

В машинном обучении нейронная сеть представляет собой модель, вдохновленную нейронной организацией, обнаруженной в биологических нейронных сетях в мозге животных.

Нейронная сеть состоит из соединенных блоков или узлов, называемых искусственными нейронами, которые в общих чертах моделируют нейроны головного мозга. Они соединены ребрами, которые моделируют синапсы в мозге. Искусственный нейрон получает сигналы от подключенных нейронов, затем обрабатывает их и отправляет сигнал другим подключенным нейронам. «Сигнал» – это действительное число, и выходной сигнал каждого нейрона вычисляется с помощью некоторой нелинейной функции от суммы его входных данных, называемой функцией активации. Нейроны и ребра обычно имеют вес, который корректируется по мере обучения. Вес увеличивает или уменьшает силу сигнала при соединении.

Обычно нейроны объединяются в слои. Разные слои могут выполнять разные преобразования на своих входных данных. Сигналы передаются от первого уровня (входного слоя) к последнему слою (выходному слою), возможно, проходя через несколько промежуточных слоев (скрытых слоев). Сеть обычно называют глубокой нейронной сетью, если она имеет по крайней мере два скрытых слоя.

Искусственные нейронные сети используются для прогнозного моделирования, адаптивного управления и других приложений, где их можно обучать с помощью набора данных. Они также используются для решения задач в искусственном интеллекте. Сети могут учиться на опыте и могут делать выводы из сложного и, казалось бы, не связанного между собой набора информации.

На сегодняшний день искусственный интеллект – это сквозная технология цифровой экономики, которая задает тенденцию современному миру. Изучение нейронных связей проще всего изучать на примере устройства головного мозга и нервной системы человека.

Нейроны в головном мозге устроены по принципу работы аксонного тела (рис. 2.1). На аксонном теле имеется сома нейрона (оболочка), от которой отходят «отростки», называемые дендритами. Дендриты в свою очередь служат в данной схеме входами, т. е. через них собирается информация из окружающего мира. Далее поступившая информация концентрируется на соме нейрона, что в свою очередь вызывает возбуждение.

Возбуждение аккумулирует в аксонном холмике (рис. 2.1.) до тех пор, пока не достигнет предельных значений, которые вызовут спуск потенциала действия. Говоря иначе, информация, проходя через дендриты (входы) должна обработаться и пройти путь по аксону т.е. к выходу.

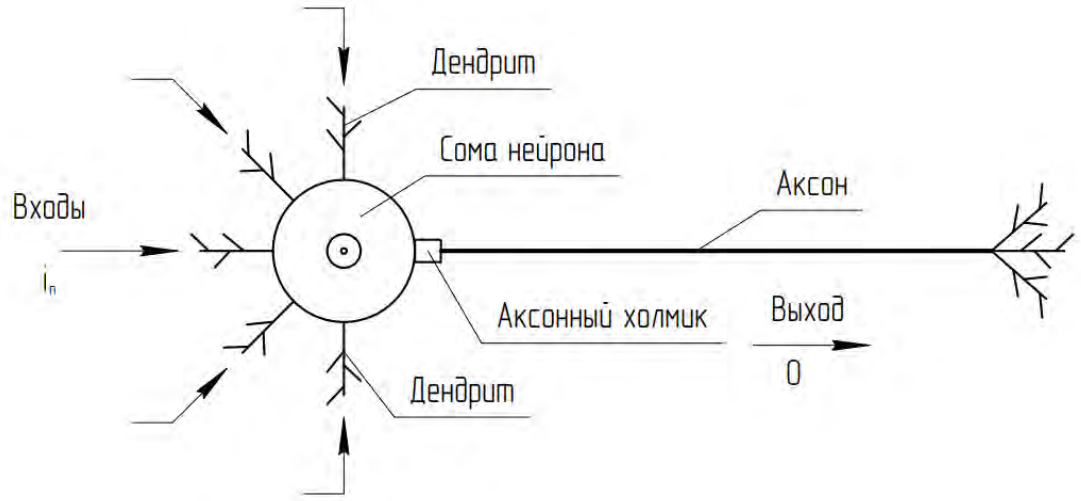


Рисунок 2.1 – Модель нейрона головного мозга

В математическом представлении данную модель можно описать следующем образом (рис. 2.2). Набор входов, представим в виде «{*I*}» – множество входов. Порог возбуждения – «*T*», который будет описываться функцией возбуждения «F_T». Так же введем понятие «функции суммирования входов – «F_Σ». На выходе получим значение «о».

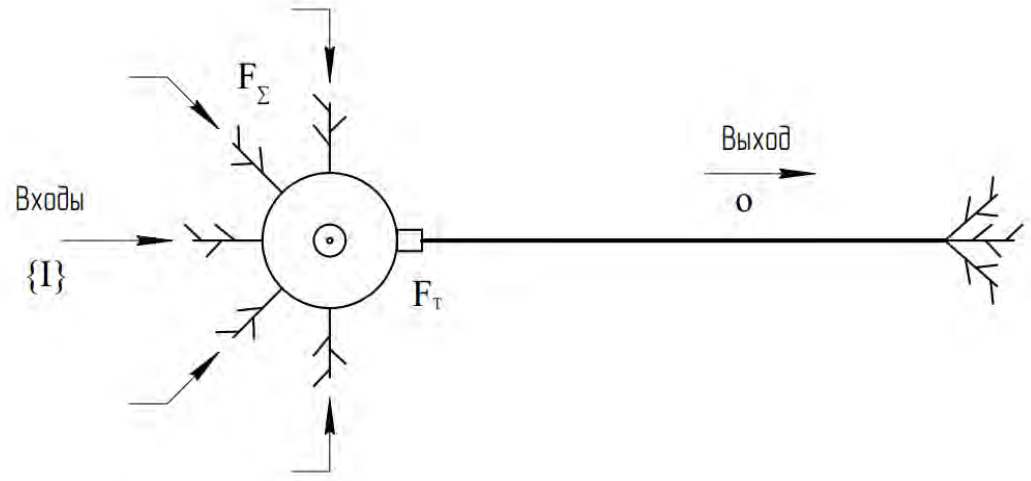


Рисунок 2.2 – Математические обозначение модели нейрона

Изначально множество входов можно разделить на два вида: тормозные и возбуждающие, которые будут выдавать значения единицы и ноля соответственно (биты). В данном представлении (возбуждающих и тормозных входов) обучение нейросети не предстоит возможным, поэтому на вход будут подаваться биты «0» или «1» (рис. 2.3), но при этом

каждый вход будет ассоциироваться и иметь соответствующий индекс числу по порядку «0 – ω_0; 1 – ω_1», что в свою очередь даст нам множество весов (входов) - «{I}».

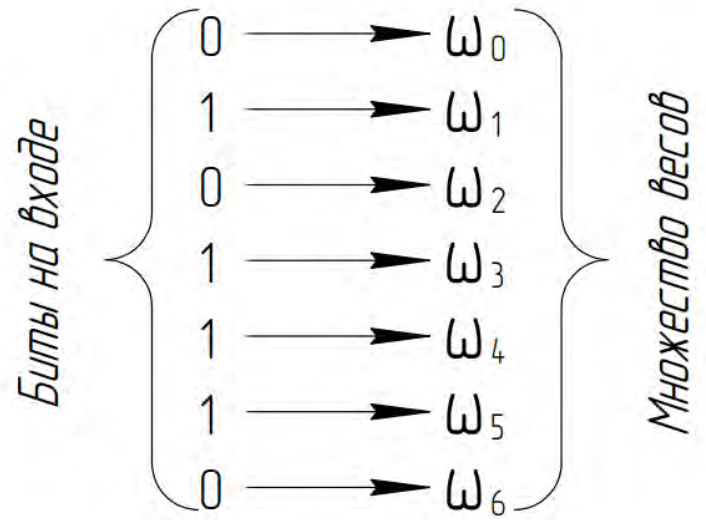


Рисунок 2.3 – Ассоциация битов на входе и множества весов

Множества весов попадают на сому нейрона и суммируются в дальнейшем. (Рис. 2.4), сома суммирует и результат передается в функцию активации, которая выдает выходное значение.

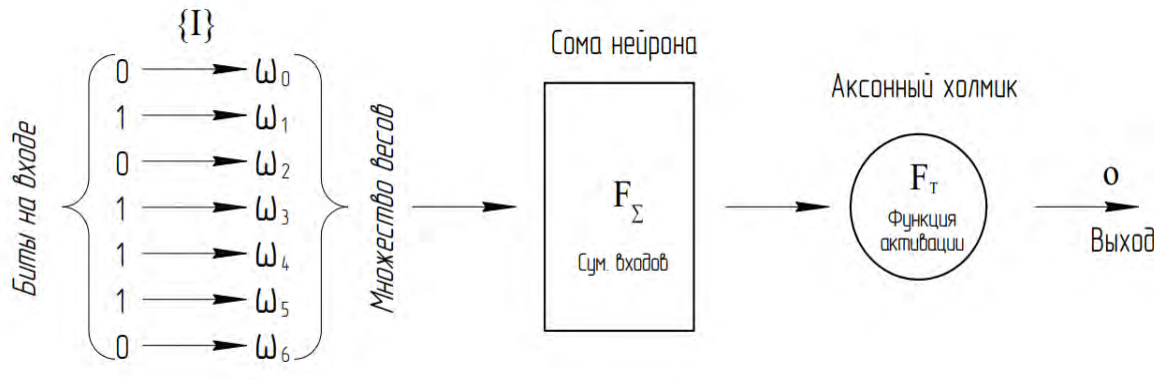


Рисунок 2.4 – Математическая модель нейрона

В данной модели, выход «o» может принимать значения либо «0» либо «1» то есть o ϵ {0; 1} бит.

Следящим этапом необходимо определить веса: ω_i, где i – может принимать значения от нуля до n – число входов $\omega_i \in [-1;+1]$.

Множество весов принадлежит интервалу от -1 до +1, т.е. если на выход и вход биты принадлежат множеству, то веса могут принимать значения из интервала.

Функция суммирования от входных векторов и вектора весов, не что иное, как сумма по i от 1 до N, т.е. происходит умножения входа на соответствующий вес:

$$F(\vec{i},\vec{\omega}) = \sum_{i=1}^{n} i_j, \omega_j. \qquad (2.1)$$

При умножении вектора входа на вектор соответствующего веса и дальнейшего суммирования, суммируются только те значения, которые больше 0 (рис. 2.5).

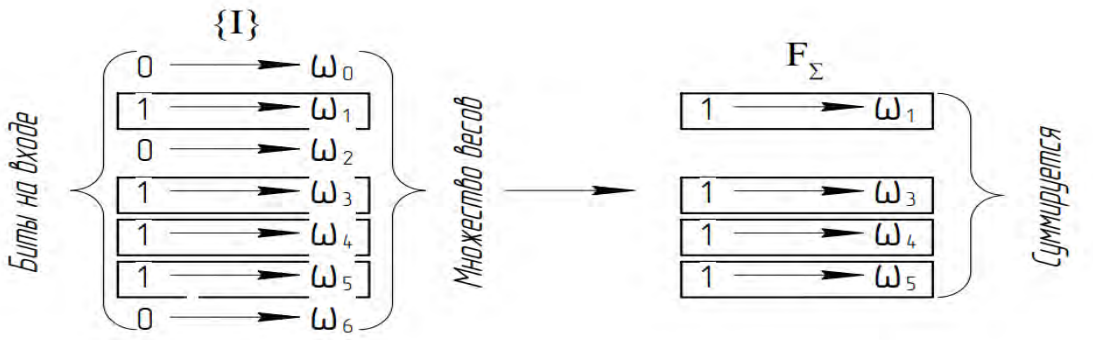


Рисунок 2.5 – Функция суммирования

Функция активации получает значение от функции суммирования и если значение, которое пришло на вход отрицательное, то функция возвращает ноль, а если аргумент положительный, то функция возвращает единицу (рис. 2.6).

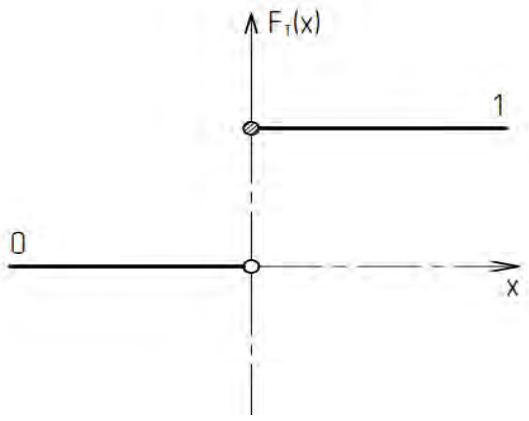


Рисунок 2.6 – График функции активации (утолщенная линия – функция активации)

То есть, если у нас есть некое значение, полученное с выхода сумматора, мы передаем его в функцию активации и функция активации возбуждает выходное значение, то есть возвращает в единицу тогда и только тогда, когда входное значение у нее равно или больше нуля.

Важным моментом является то, что функция активации не является линейной, иначе функция активации *F(X)* возвращала бы значение *X*, а это в свою очередь приводит к тому, что важные свойства искусственных нейронных сетей, не являлись бы таковыми.

Математическая модель нейрона представляет собой две функции: функцию суммирования и функцию активации. Для функции суммирования на вход подается множество входных значений, которые поступают в нейрон, и множество весов на этих значениях. Функция суммирования суммирует те веса, которым соответствуют единицы на входе, то есть установленные биты. Если к какому-то весу бит не установлен, пришел ноль, то этот вес не суммируется в общей сумме.

Фактически веса могут принимать значение от -1 до 1, действительное значение из этого интервала и функция суммирования складывает те веса, которые активированы. На выходе этой функции появляется сумма (значение). Эта сумма переходит на вход функции активации. Она возвращает ноль, если ее аргумент меньше нуля, и возвращает единицу, если ее аргумент больше или равен нулю.

На данном принципе основана математическая модель нейрона, которую описали Уорен Маккаллок и Уолтер Питтс. Данная модель была изложена в пятидесятые года XIX века.

2.2. Естественные и искусственные нейронные сети

Строение искусственного нейрона (рис. 2.8) схожа со строением естественного нейрона (рис. 2.7) живого организма. Можно найти некоторые аналогии между естественным нейронам и искусственным. В естественном нейроне есть:

– тело клетки,
– дендриты, по которым нейрон из других нейронов получает сигнал;
– синапсы могут либо пропускать сигнал, либо тормозить;
– аксон, канал выходного сигнала.

Нейрон генерирует один выходной сигнал и подает его, за счет ответвлений на аксоне сразу в несколько нейронов. Эти базовые идеи реализованы в искусственном нейроне.

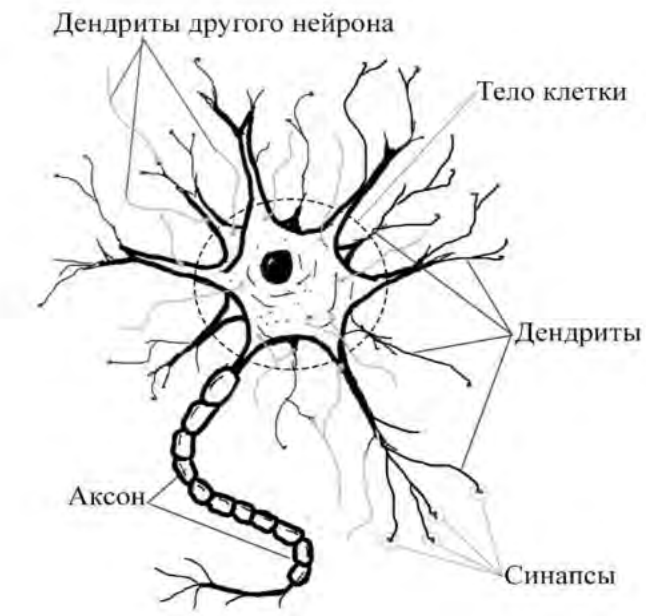


Рисунок 2.7 – Естественный нейрон

Искусственный нейрон – это вычислитель передачи данных и сумматор. В тело нейрона попадают значение извне, как в естественном нейроне через через дендриты, то есть набор чисел x_1, $x_2 \ldots x_n$ и числа умножаются на синаптические веса θ_1, $\theta_2 \ldots$ θ_n. Значения суммируется точно так же, как в задачах регрессии. Добавляется коэффициент θ_0, который поднимает и опускает графическую зависимость математической модели. В задачах логистической регрессии коэффициент θ_0, который всегда равен единице, смещает границу разделяя класс. Коэффициент θ_0 представляет собой искусственно добавленный сигнал.

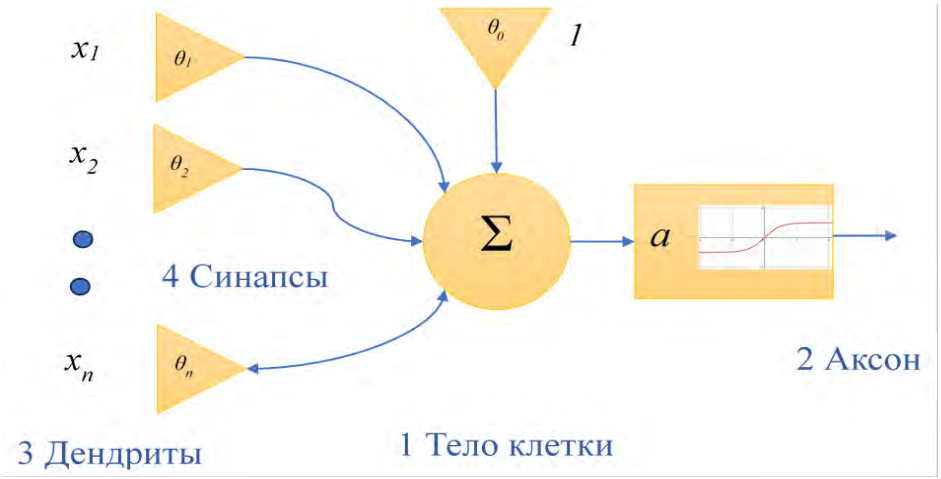


Рисунок 2.8 – Искусственный нейрон
1 – тело клетки - обрабатывает информацию;
2 – аксон - передает обработанную информацию другим нейронам;
3 – дендриты - получают информацию от других нейронов;
4 – синапсы - соединяют аксон и дендриты других нейронов

Несмотря на визуальную схожесть искусственного и естественного нейрона первый сильно отстает в производительности.

Длительность одной операции составляет:

– для искусственного нейрона – 10^{-9} с;

– для естественного – 10^{-3} с.

Архитектура нейронной сети:

– в искусственной нейронной сетей (ИНС) порядка 10^2 нейронов и 10^4 синаптических связей. Каждый нейрон связан с 10^2 соседних нейронов;

– в коре головного мозга порядка 10^9 нейронов и 10^{12} синаптических связей. Каждый нейрон связан с 10^4 соседних нейронов.

Энергозатраты на выполнение 1 операции в секунду:

– для искусственного нейрона – 10^{-6} Дж;

– для естественного – 10^{-16} Дж.

Нейроны объединяются в нейронную сеть (рис. 2.9).

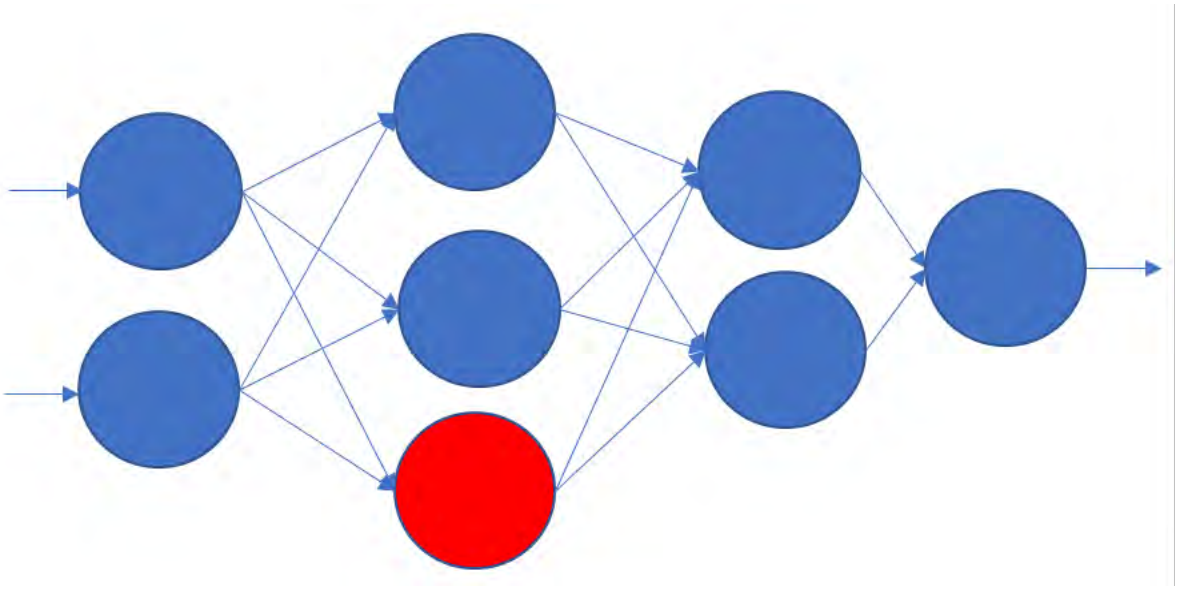


Рисунок 2.9 – Визуализация нейронной сети

В нейронной сети выполняется две основных операции:

– первая основная операция суммирования всех входов и умноженное на синаптические веса;

– вторая операция – это применение функция активации, после того как произведено суммирование и умножения на веса применяется функция активации.

Существуют следующие функции активации.

Функция тождества (Identity) $a = z$, график функции представлен на рисунке 2.10.

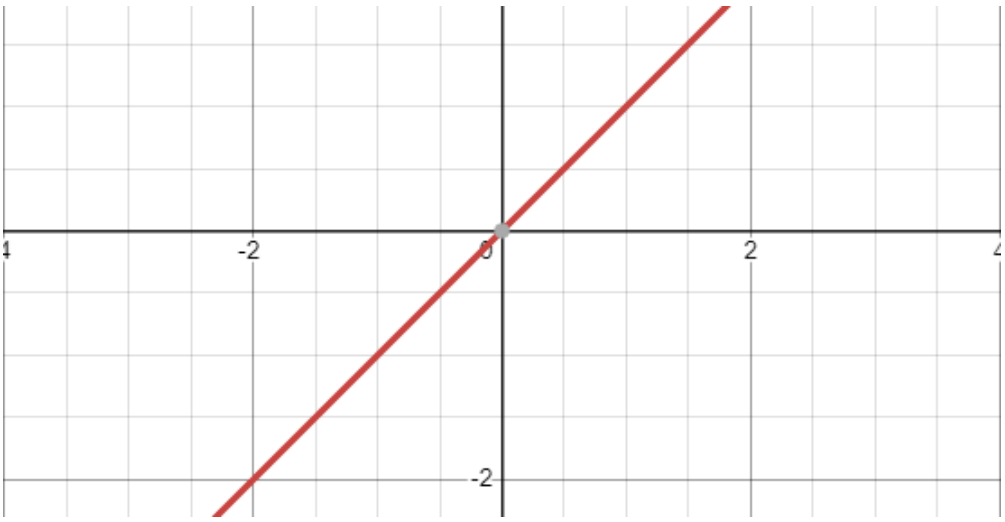


Рисунок 2.10 – График функции Identity

В математике функция идентификации, также называемая отношением идентичности, является отображением идентичности или преобразованием идентичности, представляет собой функцию, которая всегда возвращает значение, которое использовалось в качестве ее аргумента, без изменений. То есть, когда *f* является функцией идентификации, то равенство *f(x)* = *x* верно для всех значений *x*, к которым может быть применено *f*.

Сигмоидальная функция, а именно, логистическая функция активации (Logistic) $\alpha = \frac{1}{1+e^{-z}}$, график функции представлен на рисунке 2.11.

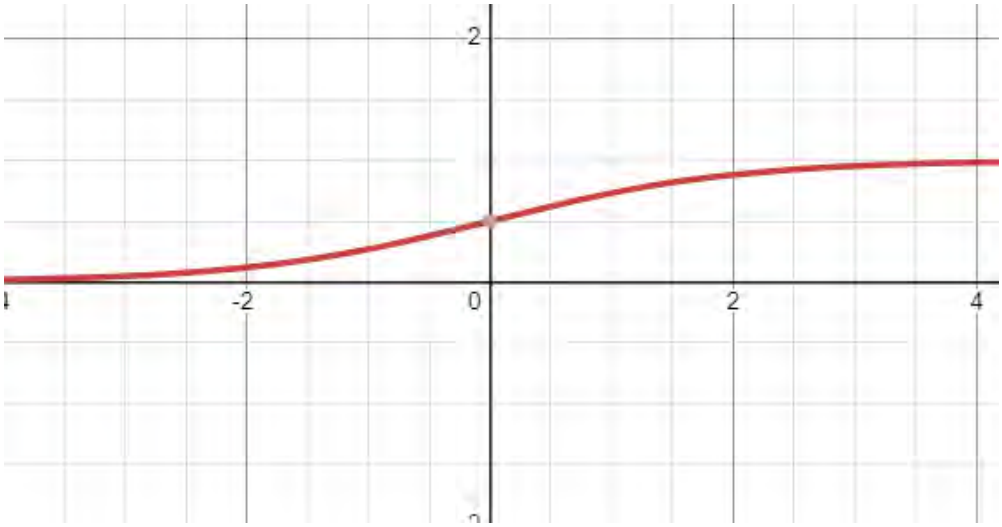


Рисунок 2.11 – График функции Logistic

Логистическая функция или логистическая кривая моделирует *S*-образную кривую роста некоторого множества *P*. Начальная стадия роста примерно экспоненциальная; затем, когда начинается насыщение, рост замедляется, а при достижении зрелости рост прекращается.

Гиперболический тангенс (Hyperbolic tangent) $\alpha = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$, график функции представлен на рисунке 2.12;

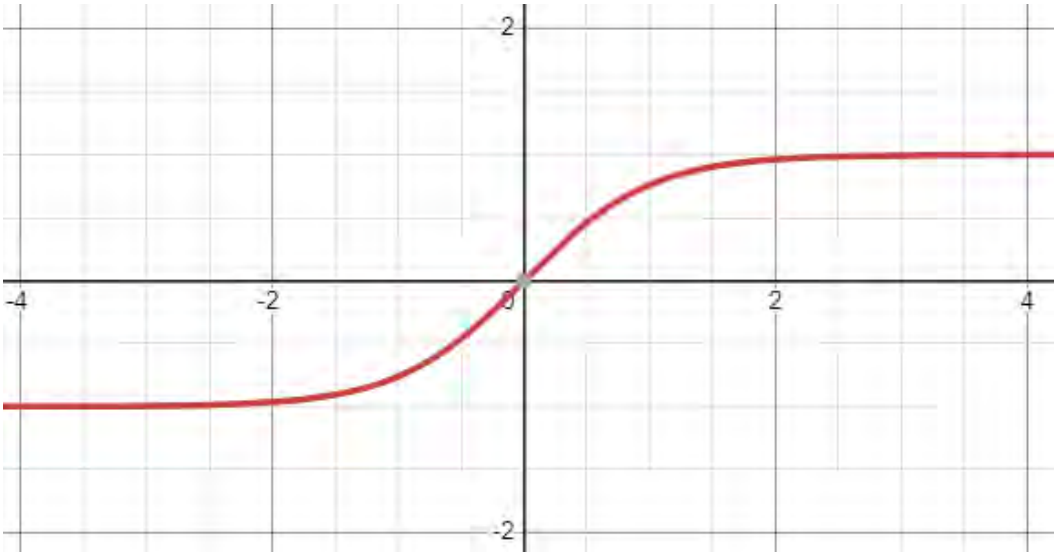


Рисунок 2.12 – График функции Hyperbolic tangent

В математике гиперболические функции являются аналогами обычных тригонометрических функций, но определяются с использованием гиперболы, а не окружности. Точно так же, как точки (*cos t*, *sin t*) образуют окружность единичного радиуса, точки (*cosh t*, *sinh t*) образуют правую половину единичной гиперболы. Кроме того, аналогично тому, как производные от *sin(t)* и *cos(t)*, являются *cos(t)* и *–sin(t)* соответственно, производными от *sinh(t)* и *cosh(t)*, являются *cosh(t)* и *+sinh(t)* соответственно.

Softplus $\alpha = ln(1+e^{z})$, график функции представлен на рисунке 2.13.

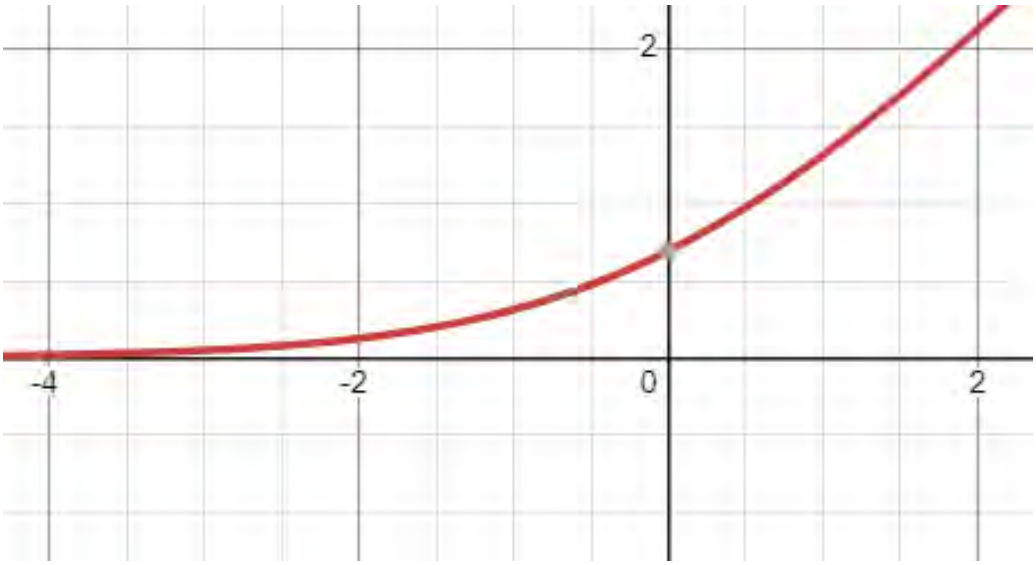


Рисунок 2.13 – График функции Softplus

Функция Softplus является непрерывной аппроксимацией ReLU. Softplus является непрерывным и может обладать хорошими свойствами с точки зрения выводимости. Интересно использовать его, когда значения находятся в диапазоне от 0 до 1. Как правило, проблематично, когда имеется много отрицательных значений, поскольку результат становится действительно близким к 0 и может привести к гибели нейрона.

Полулинейная функция (Rectified linearn Unit (ReLU)) α=max(0;z) график функции представлен на рисунке 2.14.

В контексте искусственных нейронных сетей функция активации выпрямителя или ReLU (выпрямленной линейной единицы) – это функция активации, определяемая как положительная часть ее аргумента. Функция работает аналогично полуволновому выпрямлению в электротехнике.

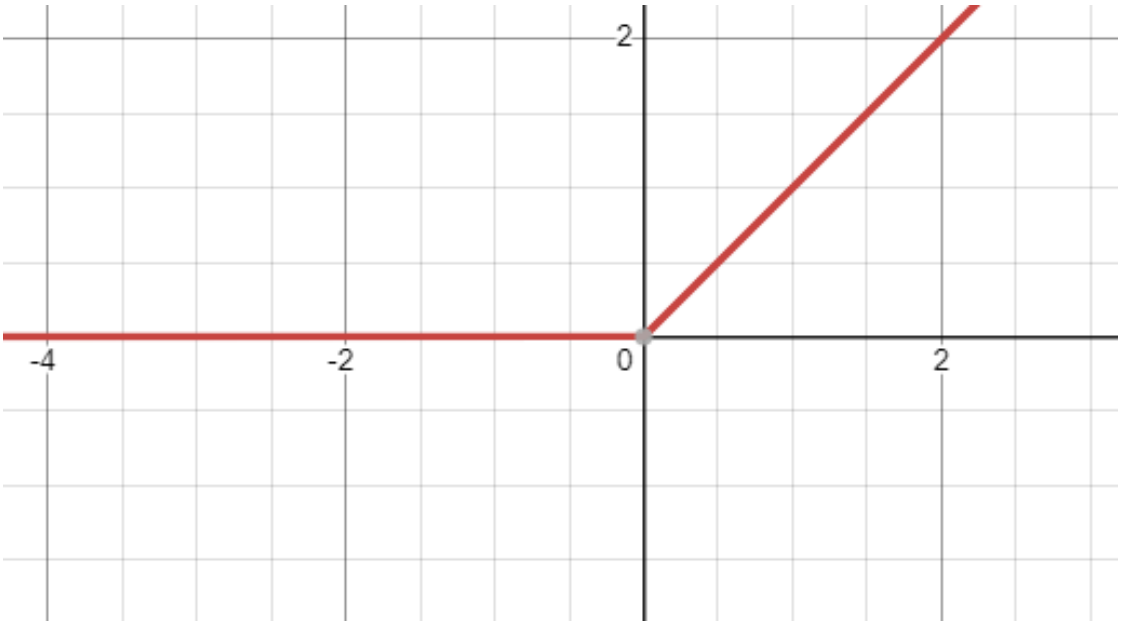


Рисунок 2.14 – График функции Rectified linearn Unit

Функция активации softmax $\alpha_j = \frac{e^{z_j}}{\sum_j e^{z_j}}$ преобразует исходные данные нейронной сети в вектор вероятностей, распределение вероятностей по входным классам. Используется если несколько выходных значений, обычно применяется в выходном слое нейронной сети.

Архитектура искусственной нейронной сетей (ИНС) прямого распространения показана на рисунке 2.15.

Входной слой: каждый нейрон имеет ровно один вход от внешней среды. Нейроны что получают то и выдают. Нейрон входного слоя имеет всего один входной сигнал, то есть одно число какой-то извне поступает на один нейрон.

Скрытый(ые) слой(и): в каждый нейрон следующего слоя поступает сигнал от каждого нейрона предыдущего слоя. Скрытых слоев может быть 0,1…*n*. В решении задач аппроксимации достаточно 1 скрытого слоя.

Выходной слой: каждый нейрон может иметь много входов, но один выход, который является реакцией сети.

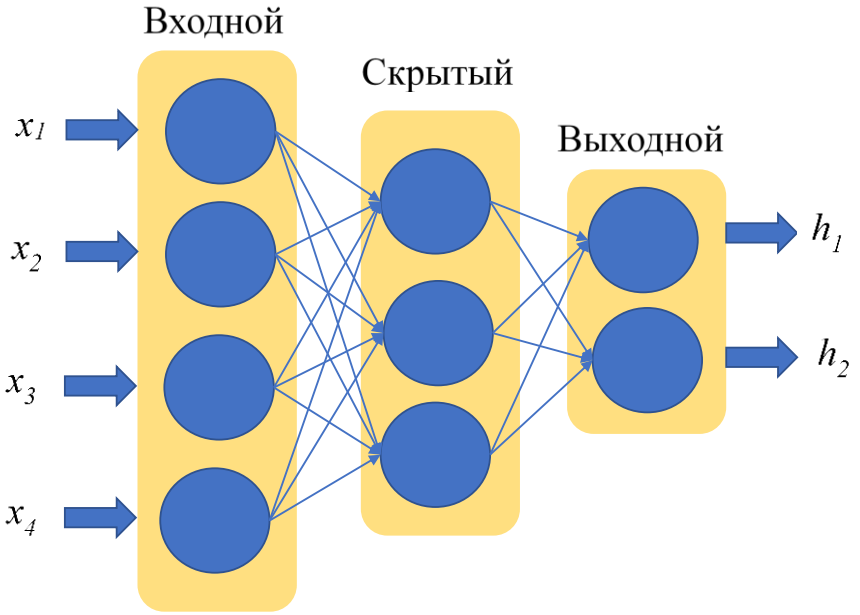


Рисунок 2.15 – Архитектура сетей прямого распространения

Структура полно связной нейронной сети представлены рисунке 2.16.

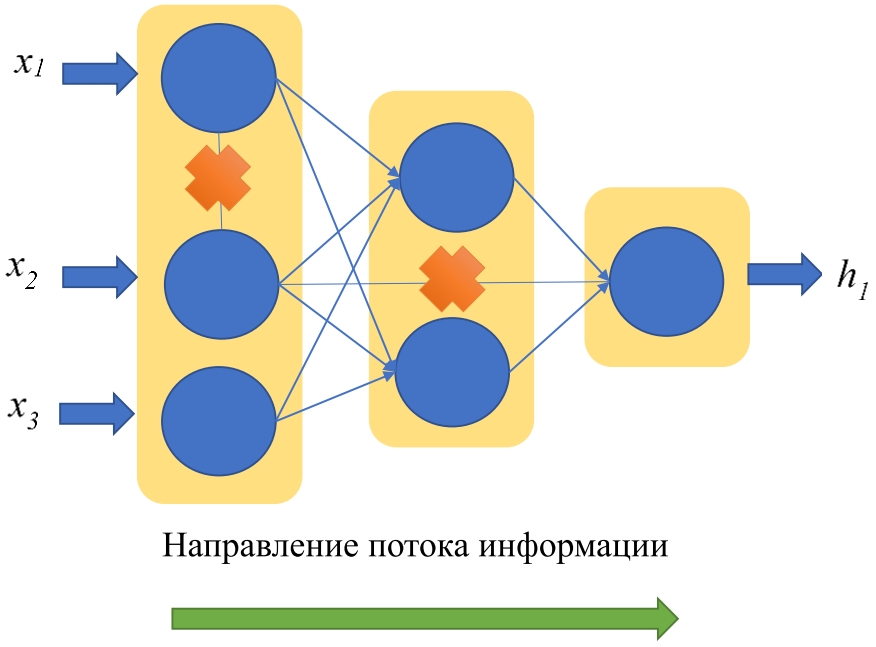


Рисунок 2.16 – Полносвязная нейронная сеть

Правила строения полносвязной нейронной сети:

– каждый нейрон следующего слоя связан со всеми нейронами предыдущего;

– нейроны слоя не связаны;

– нейроны передают информацию только нейронам следующего слоя;

– перепрыгивание через слои запрещено.

Для настройки сети необходимо выбрать: количество входных величин x_i и количество нейронов в выходном слое h.

Задаваемые и не варьируемые параметры нейронной сети:

– количество входных нейронов (из условия задачи);

– количество скрытых слоев (задается программистом);

– количество нейронов в скрытых слоях (задается программистом);

– количество выходных нейронов (из условия задачи);

– функция активации нейронов (задается программистом).

В процессе обучения варьируются веса каждого соединения для того, чтобы ответ в последствии был наиболее точным.

2.3. Прямые вычисления в искусственной нейронной сети

Обучение с учителем подразумевает наличие правильных ответов (labeled data), которые можно сравнить с результатами вычислений ИНС.

Задача распознавания (классификации) рукописных чисел.

Архитектура искусственной нейронной сетей включает в себя: количество слоев – l; количество нейронов k-ом слое – n_k (n_l – количество классов); логистическая функция активации в скрытых слоях и функция активации «softmax» в выходном слое.

При рассмотрении задачи классификации цифр от 0 до 9 используется алгоритм представленный на рисунке 2.17.

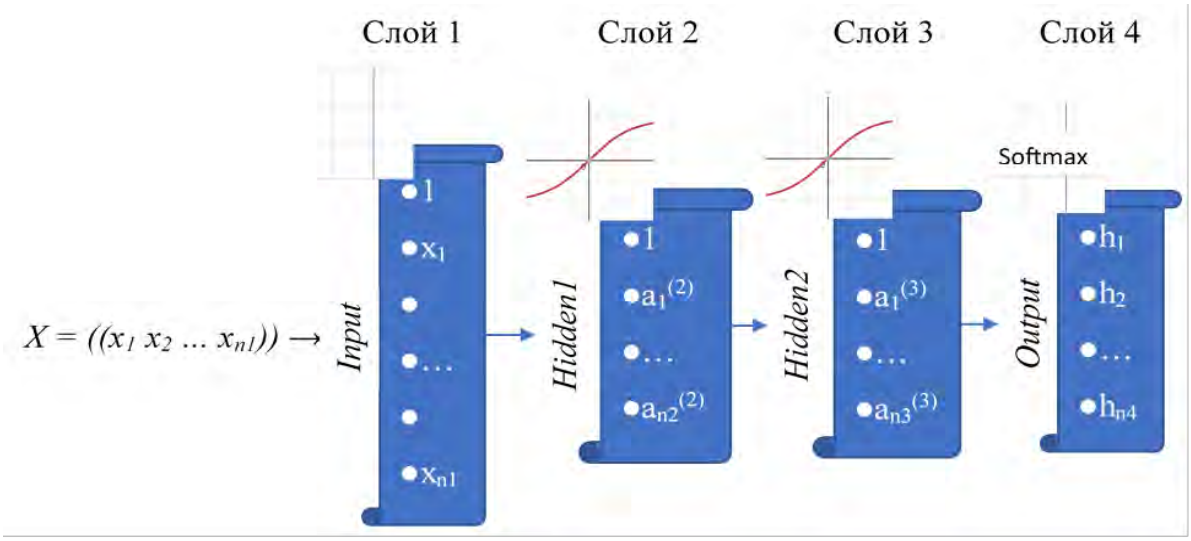


Рисунок 2.17 – Визуализация задачи

Если на вход нервной сети подается изображение, то изображение разбивается на пиксели и подается в виде набора чисел. Числа можно представить в виде матрицы строки, все они подаются на вход.

По стандартной процедуре известной из логистической и линейной регрессии добавляется один нейрон, который равен единице.

Вычисления в прямом направлении ИНС, производятся путём расчета матриц $A(k)$ результатов в каждом слое.

Слой 1 (входной). На вход слоя подается дополненная единицей матрица X. Активация здесь нет, поэтому на выходе из этого слоя то же самое что и на входе, но дополненная единицей: $A^{(1)} = ((1 \quad X))$.

Слой 2 (скрытый). Данные с первого слоя умножается на веса $\Theta^{(1)}$ и суммируется: $Z^{(2)} = A^{(1)} \Theta^{(1)}$. Затем применяется функция активации $A^{(2)} = ((1 \; sigmoid(Z^{(2)})))$. При этом на данной слое снова добавляется один нейрон равный единице.

Слой 3 (скрытый). Процесс повторяется: то, что получилось после активации во втором слое умножается на веса второго слоя $\Theta^{(2)}$: $Z^{(3)} = A^{(2)} \Theta^{(2)}$, выходом с третьего слоя является применение сигмоиды $A^{(3)} = ((1 \; sigmoid(Z^{(3)})))$.

Слой 4 (выходной). То, что получилось после активации в третьем слое умножается на веса третьего слоя $\Theta^{(3)}$: $Z^{(4)} = A^{(3)} \Theta^{(3)}$. Затем применяется функция активации 4-го слоя «softmax» $A^{(4)} = softmax(Z^{(4)}) = H$. Выходом этой нейронной сети является матрица гипотез в которой указаны вероятности ответ равен какому-то числу. В матрице гипотез указаны вероятности и элемент в этой матрице с наиболее близким к единице значением как правило является верным ответом.

Вычисления в задаче логистической регрессии и бинарной классификации.

Так же можно провести аналогию с логистической регрессией и бинарной классификацией, когда необходимо определить ответ сети либо ноль, либо единица:

На вход подается матрица X:

$$X = \begin{pmatrix} \begin{pmatrix} x_1^{(1)} & x_2^{(1)} \\ \dots & \\ x_1^{(m)} & x_2^{(m)} \end{pmatrix} \end{pmatrix}. \tag{2.2}$$

Правильными ответами является матрица Y:

$$Y = \begin{pmatrix} \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix} \end{pmatrix} \tag{2.3}$$

Матрица X дополняется единицей:

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{pmatrix}. \tag{2.4}$$

Дополненная матрица X умножается на матрицу весов Θ:

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{pmatrix}. \tag{2.5}$$

После дополненная матрица X умножается на матрицу весов Θ и применяется логистическая функция активации:

$$Z = X\theta \rightarrow h(z) = \frac{1}{1+e^{-z}}. \tag{2.6}$$

В искусственной нейронной сети происходят те же действия, но разница заключается в том, что значения матрицы X подаются по очереди.

Так же рассмотрим все вышеперечисленные действия в скалярном виде:

Слой первый, где задаются исходные данные матрица X и желаемый конечный результат Y матрица $X,Y \rightarrow A^{(1)} = ((1\ x_1 \dots x_{n1}))$, $A^{(1)}$ является функцией активации перового слоя (например, значения цветов всех пикселей картинки дополненные единицей).

Слой второй, в котором полученные данные из первого слоя умножаются на матрицу коэффициента веса $\theta^{(1)}$ для второго слоя и суммируется:

$$\theta^{(1)} = \begin{pmatrix} \theta_{01}^{(1)} & \theta_{02}^{(1)} & \dots & \theta_{0n_1}^{(1)} \\ \theta_{11}^{(1)} & \theta_{12}^{(1)} & \dots & \theta_{0n_2}^{(1)} \\ \dots & \dots & \dots & \dots \\ \theta_{n_1 1}^{(1)} & \theta_{n_1 2}^{(1)} & \dots & \theta_{n_1 n_2}^{(1)} \end{pmatrix}. \tag{2.7}$$

Значения в матрице весов характеризуется тремя индексами $\Theta_{n_1 n_2}^{(1)}$:

(1) – номер слоя

n_1 – характеризует нейрон текущего слоя

n_2 – характеризует нейрон следующего слоя

Расчет суммирования умножения на веса в матричном и скалярном виде соответственно:

$$Z^{(2)} = A^{(1)}\theta^{(1)}. \tag{2.8}$$

$$z_j^{(2)} = a_i^{(1)} \theta_{ij}^{(1)} . \quad (2.9)$$

Затем применяется функция активации сигмоиды в которую добавляется нейрон с единицей в матричном и скалярном виде соответственно:

$$A^{(2)} = ((1 sigmid(Z^{(2)}))) . \quad (2.10)$$

$$a_0^{(2)} = 1, a_j^{(2)} = \frac{1}{1 + e^{-z_j^{(2)}}}, (j = 1,...n_2). \quad (2.11)$$

В выходном слое расчёт функция активации происходит с применением «softmax» в матричном и скалярном виде соответственно:

$$A^{(4)} = soft \max(Z^{(4)}) \quad . \quad (2.12)$$

$$a_j^{(4)} = \frac{e^{z_j^{(4)}}}{\sum_{i=1}^{(n_4+1)} e^{z_j^{(4)}}} . \quad (2.13)$$

Если известен результат расчета, далее необходимо нейронную сеть обучить. Для обучения нейронной сети необходимо её охарактеризовать, то есть выяснить на сколько ответы нейронной сети близки к правильному ответу.

Для этого производится расчёт функции качества. Расчёт функции качества в задаче логистической регрессии выполняется по формуле 1.27. Расчёт производится по всем экспериментальным данным.

Функция качества в ИНС отличается, так как имеет несколько гипотез из-за того, что необходимо просчитать ошибку для каждой единицы исходной информации.

Функция качества в ИНС. Количество слоев – l; количество нейронов k-ом слое – n_k (n_l – количество классов); логистическая функция активации в скрытых слоях и функция активации «softmax» в выходном слое.

Функция качества для ИНС выглядит следующим образом:

$$J(\theta^{(k)}) = -\frac{1}{m} \sum_{i=0}^{m} \sum_{j=1}^{n_1} (y_j^{(i)} \ln(h_j^{(i)}) + (1 - y_j^{(i)})(\ln(1 - h_j^{(i)})) + \frac{\lambda}{2m} \sum_{k=1}^{i-1} \sum_{i=1}^{n_k} \sum_{j=1}^{n_{k+1}} (\theta_{ij}^{(k)})^2 \Rightarrow \min \quad (2.14)$$

Функцию ошибки необходимо считать для всего набора данных. Также добавляется регуляризирующее слагаемое.

2.4. Обратные вычисления в искусственной нейронной сети

Обратные вычисления в искусственной нейронной сети – это вычисления которые необходимы для подбора коэффициентов (для обучения), аналогично в регрессионном анализе реализовано с помощью метода градиентного спуска.

Обратный расчет выполняется с целью определения компонент градиента функции качества и является этапом процесса обучения.

Алгоритм выполнения обратных вычислений следующий: имеется функция ошибки $L(\theta^{(k)})$, зависящая от матрицы весов θ и вектор градиента – производная функции качества L по всем возможным весам θ $[\![\partial L / \partial \theta_{ij}^{(k)}]\!]$. Далее с помощью метода обратного распространения ошибки определяется новые значения весов и новые значения функции качества и функция зацикливается $\theta_{ij}^{(k)} \to L(\theta^{(k)}) \to \ldots$.

Пример. ИНС с двумя входными нейронами x_1, x_2 и одним выходным нейроном h который надо сравнить со значением y. Во вход слое функция активации будет всегда линейная. В представленном примере скрытые слои отсутствуют. В выходном слое функция активации логистическая и данную функцию необходимо определить для набора значений базы данных m штук для матриц X (рис 2.18).

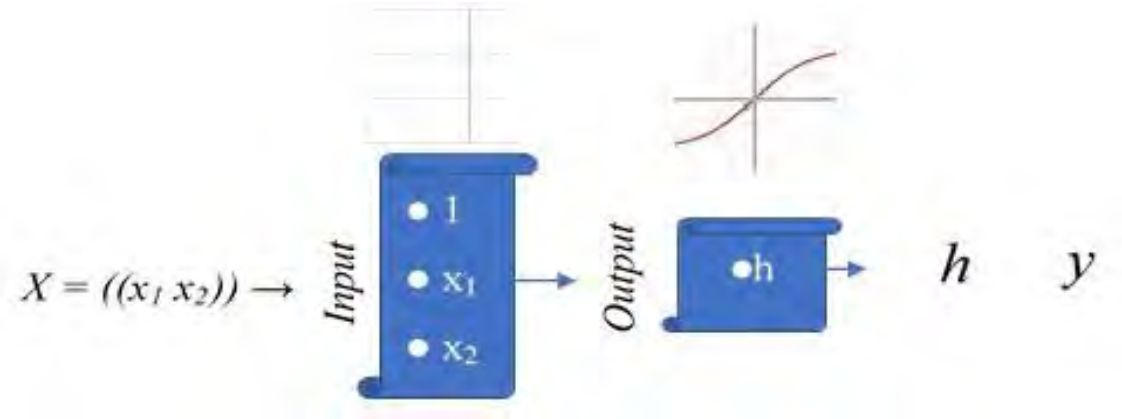


Рисунок 2.18 – Визуализация задачи

Функция активации представлена в уравнении 2.15:

$$L(\theta^{(1)}) = -\sum_{i=1}^{m} (y^{(i)} \ln(h^{(i)}) = -\sum_{i=1}^{m} \ln(h^{(i)}) \Rightarrow \min \qquad (2.15)$$

Функция активации (уравнение 2.15) усечена, в сравнении предыдущей рассматриваемой функцией, исключено одно слагаемое. Но на практике достаточно для применения представленной функции активации. Зна-

чения *y* в бинарной классификации представляют 0 или 1. Нулевую компоненту можно не учитывать на функцию ошибки не окажет влияния. Если *y* равен единице, то можно не записывать значения в уравнении. Соответственно функцию качества для dataset из *m*-штук будет иметь конечный вид, представленный в правой части уравнения 2.15. Данную функцию необходимо минимизировать.

Алгоритм обратного распространения ошибки. Вычисления в нейронной сети можно представить в виде графа (рис. 2.19). Так как в рассматриваемом примере простая нейронная сеть, то и граф имеет простой вид.

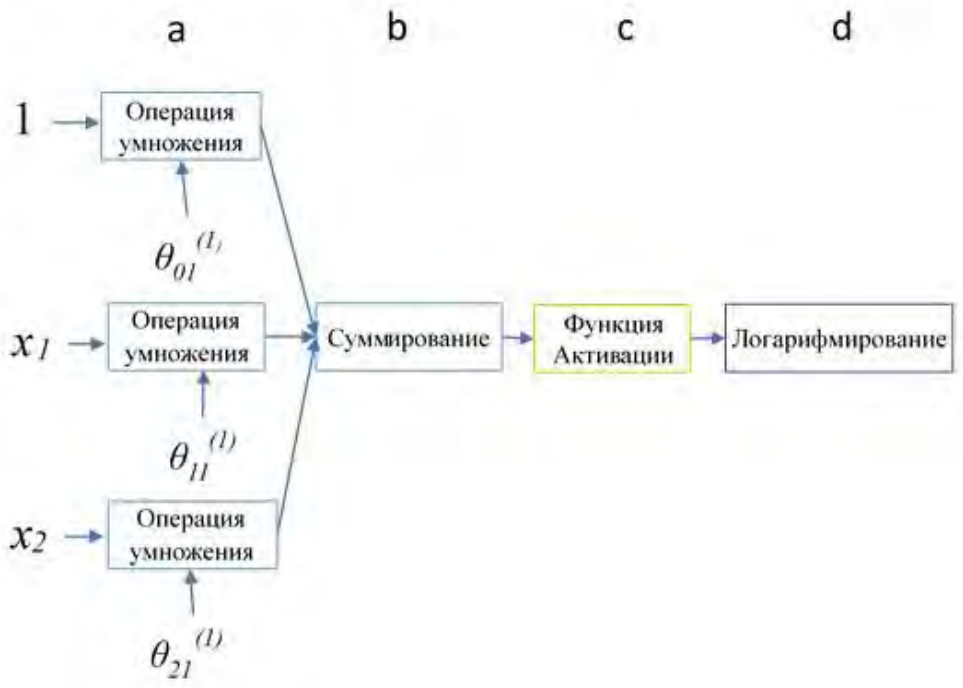


Рисунок 2.19 – Визуализация процесса работы нейронной сети

Нейрон входного слоя дополняется нулевым нейроном, сигнал которого – единица. Далее все входные значения умножаются на веса, в результате получается матрица активации первого слоя. В графах первого слоя указана операция умножения. Далее производится суммирование, и определятся матрица *Z*. На следующем этапе определяется логистическая функция активации. После применения функции активации получается ответ сети – гипотеза и к данной гипотезе применяется логарифмирование.

Пусть *m*=1, то есть в базе данных один элемент (одна пара значений x_1, x_2). По результатам вычислений определяют значение *L*. Необходимо найти одну компоненту вектора градиента $\partial L / \partial \theta_{11}^{(1)}$.

Необходимо выполнить следующие вычисления:

а. входные данные умножаются на веса $a = \theta_{11}^{(1)} x_1$, производная $\partial a / \partial \theta_{11}^{(1)} = x_1$;

b. полученные ранее значения суммируются $b = \theta_{01}^{(1)} + a + \theta_{21}^{(1)} x_2$; производная $\partial b / \partial a = 1$;

c. применяется логистическая функция активации $c = \frac{1}{1 + e^{-b}}$; производная $\partial c / \partial b = c(1 - c)$;

d. После применения функции активации получается ответ сети (гипотеза) и к этой гипотезе применяется функция логарифмирования $d = -\ln(c)$,на этом этапе d=L выдаётся как результат. Производная $\partial d / \partial c = -1 / c$ и $\partial L / \partial d = 1$.

По сути L – сложная функция «матрешка», которую можно представить как:

$$L = L\left(d\left(c\left(b\left(a\left(\theta_{11}^{(1)}\right)\right)\right)\right)\right) \tag{2.16}$$

L зависит от d, которая зависит от c, которая зависит от b, которая зависит от a, которая зависит от $\theta_{11}^{(1)}$ по которой определяется производная.

Производная $\partial L / \partial \theta_{11}^{(1)}$ первого слоя будет равна:

$$\frac{\partial L}{\partial \theta_{11}^{(1)}} = \frac{\partial d}{\partial L} \cdot \frac{\partial d}{\partial c} \cdot \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a} \cdot \frac{\partial a}{\partial \theta_{11}^{(1)}} \tag{2.17}$$

Используя значения производных выражение 2.17 примет вид:

$$\frac{\partial L}{\partial \theta_{11}^{(1)}} = 1\left(-\frac{1}{c}\right)c(1-c)1 \cdot x_1 = (c-1) \cdot x_1. \tag{2.18}$$

В случае m>1, то:

$$\frac{\partial L}{\partial \theta_{11}^{(1)}} = \sum_{i=1}^{m} (h^{(i)} - 1) x_1^{(i)}. \tag{2.19}$$

Алгоритм обучения (обобщенный).

1. Задать начальные значения компонент матрицы $\Theta^{(k)}$ случайным образом.

2. Рассчитать вектор градиента $\nabla L = \left[\left[\partial L / \partial \theta_{ij}^{(k)}\right]\right]$ методом обратного распространения ошибки.

3. Найти новые значения компонент Θ: $\theta_{ij}^{(k)^H} = \theta_{ij}^{(k)^C} - \alpha \frac{\partial L}{\partial \theta_{ij}^{(k)}}$.

4. Повторять пп. 2–3 до достижения минимума L: $L^H - L^C < \delta$ или количество итераций достигнет максимального значения: $N_{\text{итерации}} > N_{max}$.

5. Вывод результатов: результатом является все матрицы $\Theta^{(k)}$.

2.5. Настройка моделей машинного обучения

Параметры модели определяются в ходе решения задачи машинного обучения. Гиперпараметры задаются пользователем, как правило не единственным образом, и их значения влияют на значения искомых параметров. Настройка гиперпараметров – важный шаг в разработке моделей машинного обучения, поскольку он может значительно улучшить производительность модели при работе с новыми данными.

Для подготовки данных задаются нижеперечисленные параметры.

Масштабирование признаков / Feature Scaling.

Масштабирование объектов – это метод стандартизации независимых объектов, присутствующих в данных, в фиксированном диапазоне. Он выполняется во время предварительной обработки данных для обработки сильно изменяющихся величин, значений или единиц измерения. Если масштабирование признаков не выполняется, то алгоритм машинного обучения имеет тенденцию взвешивать большие значения, увеличивать их и рассматривать меньшие значения как меньшие значения, независимо от единицы измерения значений.

Скорость обучения α / Learning rate.

Скорость обучения – один из наиболее важных гиперпараметров, который необходимо настроить для нейронной сети для достижения лучшей производительности. Скорость обучения определяет размер шага на каждой итерации обучения при движении к оптимуму функции потерь. Значение α определяется в диапазоне от 0 до 1.

Погрешность δ и количество итераций N_{max} / Error and #of iterations.

Количество данных для градиентного метода / Batch gradient descent (GD) – Mini-Batch GD. – Stochastic GD.

Градиентный спуск – это широко используемый алгоритм итерационной оптимизации, который используется для нахождения минимума любой дифференцируемой функции. Каждый предпринятый шаг улучшает решение, приближаясь к отрицательному значению градиента функции, которая должна быть минимизирована в текущей точке. На каждой итерации вычисляется эмпирическая стоимость E функции и соответствующим образом обновляются параметры, используемые для достижения глобального минимума. Чтобы найти оптимальное решение, этот метод учитывает полный обучающий набор данных на каждой итерации, что может занять много времени и вычислительно дорого.

Визуализация процесса градиентного спуска показана на рисунке 2.20.

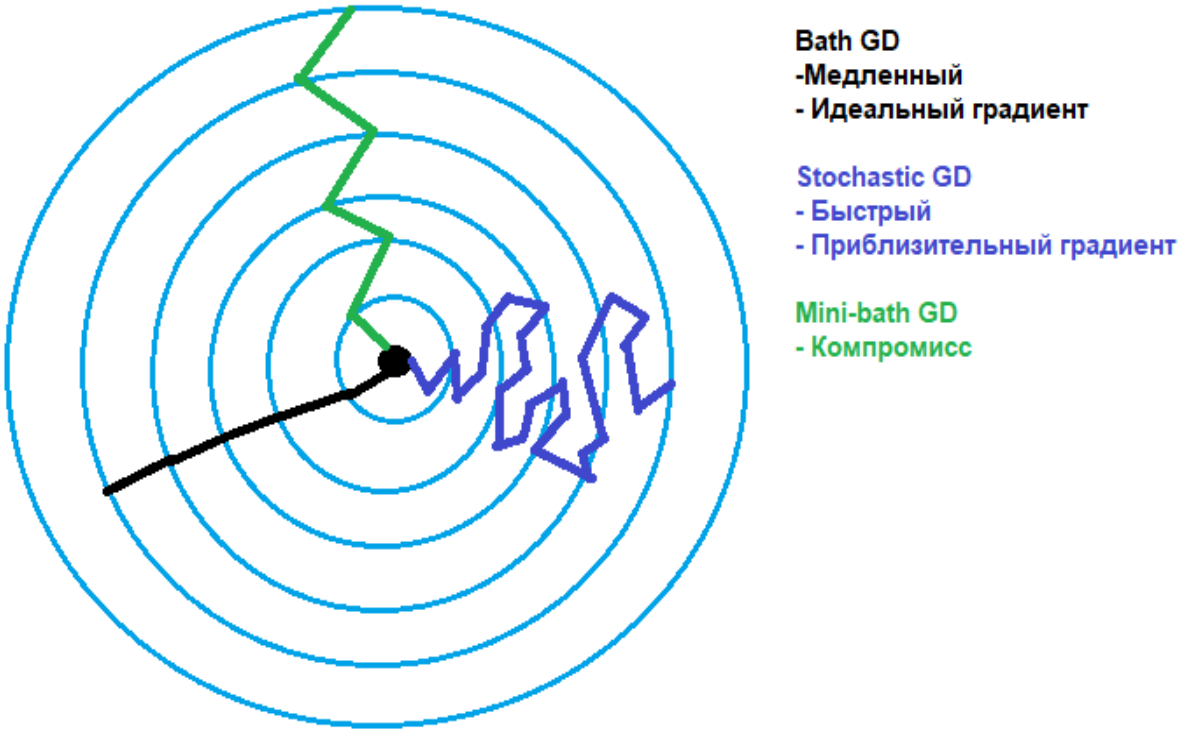


Рисунок 2.20 – Визуализация процесса градиентного спуска

Batch GD. Все данные, которые были предназначены для обучения, используются для расчета функции качества для определения градиента. Несмотря на то, что это самый гладкий самый верный путь к тому экстремум у которой необходимо найти минимум, тем не менее это процесс долгий. Проблема заключается в том, что определяется локальный экстремум, в реальных больших сетях расчет происходит непозволительно долго.

Stochastic GD. Использует всего один элемент из dataset, для посчитали функцию ошибки, определили градиент и начинается движение в его направлении, следующем шаге берется другой элемент dataset, в таком случае получается стохастический градиентный спуск. Происходит «блуждания», но данные экспериментов показывают, что можно найти экстремум и сделать это очень быстро.

Mini-Batch GD. Промежуточный вариант между двумя вышеперечисленными методами. Для обучения используется пачка элементов dataset. Из миллиона элементов берется сто и определяется значение функции, градиент, в противоположном направлении в уровне градиента делается один шаг, на следующем шаге берется случайным образом еще одна пачка элементов и все это повторяется и за большое количество шагов будет использован в итоге весь dataset. Поиск получается так же «блуждающий» из-за того, что берутся разные пачки элементов,

пространство искажается на каждом шаге, но тем не менее вот приходит к результату. Данный метод является компромиссом исходя из скорости расчета и видом градиента.

Регуляризация / Regularization.

Регуляризация в машинном обучении – это набор методов, используемых для обеспечения того, чтобы модель машинного обучения могла быть обобщена на новые данные в рамках того же набора данных. Эти методы могут помочь уменьшить влияние зашумленных данных, которые выходят за рамки ожидаемого диапазона шаблонов. Регуляризация также может улучшить модель, упростив обнаружение соответствующих крайних случаев в рамках задачи классификации.

3. ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ И ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

3.1. Обучение без учителя. Кластеризация

Задача обучения с учителем – разделение данных происходит по определённым признакам (отличиям) на кластеры. То есть на рисунке 3.1б данные размечены и есть понимание что относится к первому классу, а что ко второму и при появлении новых данных программа их классифицирует.

Обучение без учителя предполагает, что данные для нейронной сети не размечены по классам, как изображено на рисунке 3.1а.

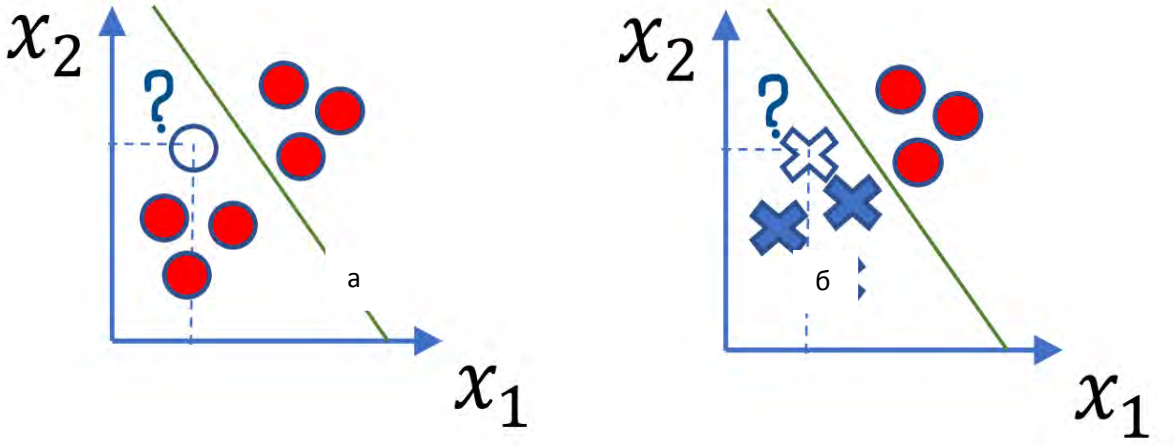


Рисунок 3.1 – Неклассифицированные данные для нейронной сети
а – Обучение без учителя; б – обучение с учителем;
зеленая линия – условная граница разделения

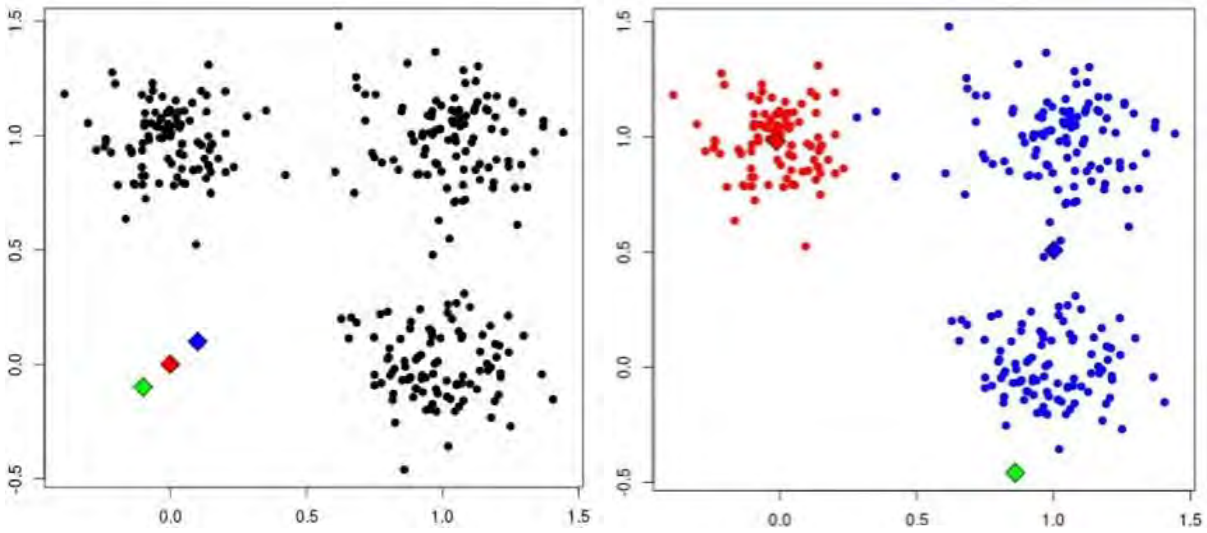
В обучении без учителя данные разделяются по отличительным признакам, как именно это происходит, рассмотрим ниже.

Кластер – сравнительно однородная группа.

Кластерный анализ представляет собой статистическую процедуру, выполняющую сбор данных, содержащих информацию о выборке объектов, и затем упорядочивающую объекты в кластеры (сравнительно однородные группы).

Среди множества методов кластерного анализа наиболее популярным является метод *k*-средних. Его основная идея заключается в разделении данных на «*k*» кластеров по признаку наименьшего расстояния до одного из центров кластеров. При этом центр кластеров пересчитывается итерационно. Количество групп (кластеров) задаётся исследователем самостоятельно.

Алгоритм кластеризации представлен на рисунке 3.2.



а б

Рисунок 3.2 – Итерационный пересчёт кластеров методом *k*-средних (кластеризация *k*-средних).
а – начальный этап кластеризации; б – итоговый результат;

Алгоритм обучения включает следующие этапы:

1. назначается количество кластеров «*k*». Количество кластеров задаётся исследователем;

2. случайным образом назначаются центры кластеров $\mu_i^{(k)}$, где параметр *i* представляет собой рациональное число от 1 до *N*. Параметр *N* определяет мерность пространства (*i=1,2,…N*).

3. определяется принадлежность каждой точки $x_i^{(j)}$, в которой параметр *i* представляет собой рациональное число от 1 до *m*, параметр *m* определяет мерность пространства, к одному из кластеров по признаку наименьшей из «*k*» величин: $\min_k \sum_{i=1}^{m} (x_i^{(j)} - \mu_i^{(k)})^2$, *m* – количество точек. При этом измеряется расстояние от конкретной точки до каждого кластера. Точка будет относится к тому кластеру до которого расстояние минимальное;

4. вычисляются новые координаты каждого кластера $\mu_i^{(k)}$ как средние арифметические координат элементов данного кластера;

5. повтор пунктов 3, 4, проводится минимизация функции *J*, т.е. минимизируется расстояние от всех точек до кластеров:

$$j = \frac{1}{m} \sum_{i=1}^{m} (x_i^{(j)} - \mu_i^{(c_i)})^2. \qquad (3.1)$$

Особенности обучения при помощи кластерного анализа.

1. Необходимость выбора количества кластеров исследователем. При недостаточности данных может возникнуть проблема с правильностью выбора количества кластеров.

Например, есть выборка данных по людям по параметру Рост-Вес. Разбиение данной выборки на группы представлено на рисунке 3.3.

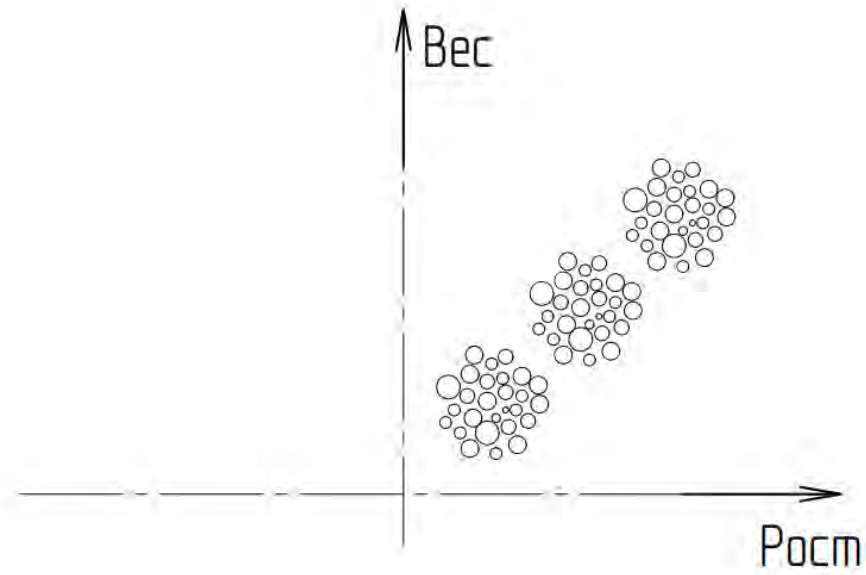


Рисунок 3.3 – Положение данных по людям на графике Рост-Вес

Исследователю необходимо среди этих данных обозначить конкретные группы людей, которые можно будет назвать кластерами. Процесс кластеризации изображён на рисунке 3.4.

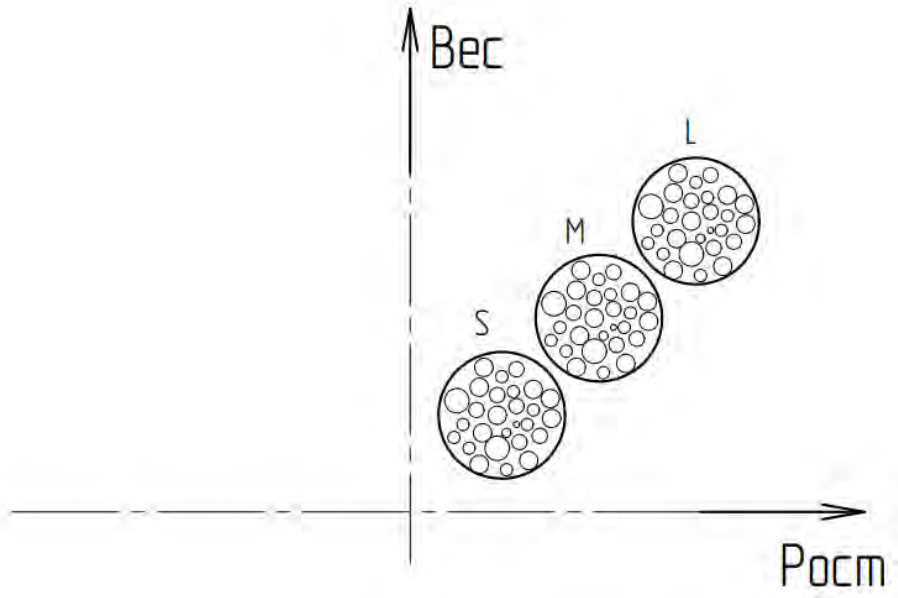


Рисунок 3.4 – Выборка данных по кластерам S, M и L

В общем виде график (рекомендация) по выбору количества кластеров представлен на рисунке 3.5.

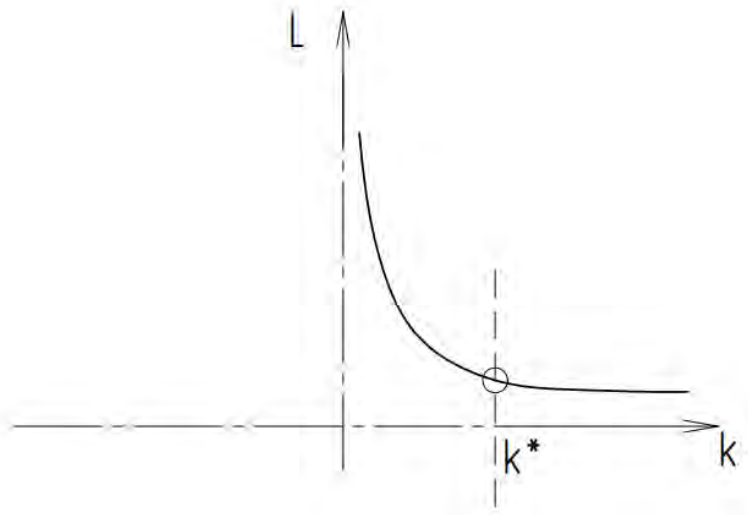


Рисунок 3.5 – График кластеризации

Принцип применения представленной зависимости следующий: число кластеров стремится к определённой точке, после которой резкий спад прекращается. Эта точка характеризует оптимальное число кластеров «*k**», необходимое для обучения нейронной сети (модели). Большое число приведёт к «деградации» нейронной сети (модели), характеризующееся возрастанием числа аномалий, меньшее число характеризует недостаточную «обученность» нейронной сети, что так же может привести к аномалиям в конечном результате.

2. Исходные данные должны иметь различимые центры кластеров. Различимость центров кластеров изображена на рисунке 3.6. По данному графику невозможно определить центры кластеров, при этом необходимо выделять 2 кластера, а при выборе метода *k*-средних граница будет являться не верной.

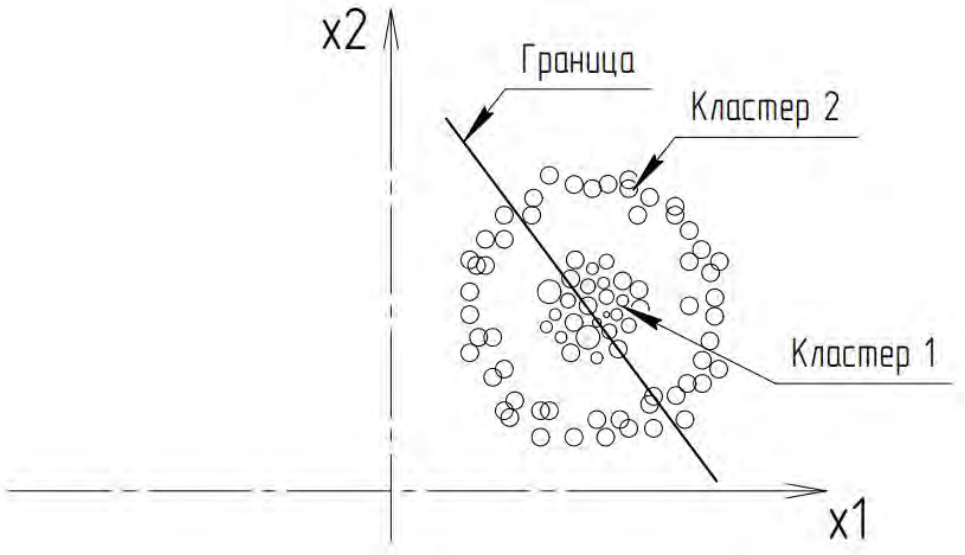


Рисунок 3.6 – График, изображающий различимый центр кластеров
и неразличимый центр кластеров (аномалия)

Определить различимость центра кластеров помогает функция нормального распределения, представлена на рисунке 3.7 (Anomaly denection).

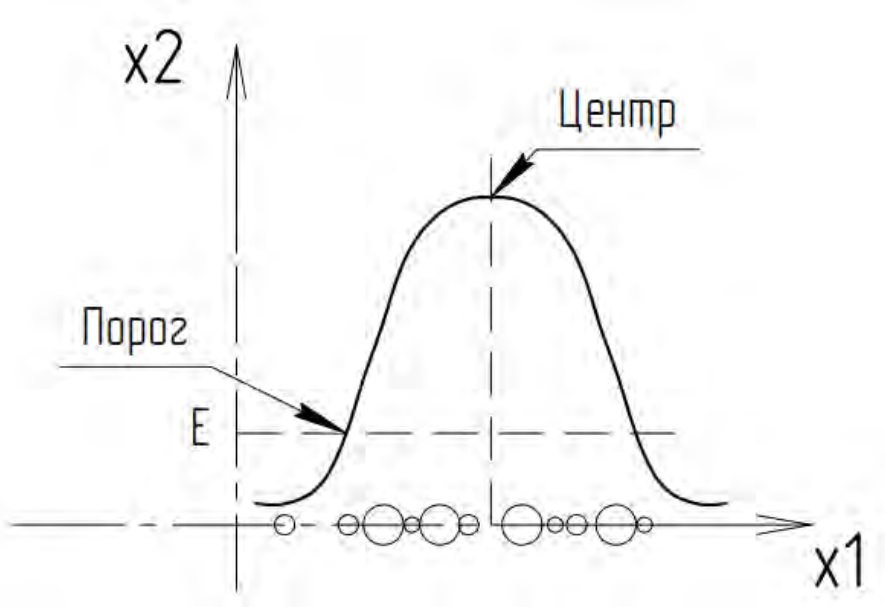


Рисунок 3.7 – График функции нормального распределения

Определяется функция нормального распределения P., а также устанавливается пороговое значение «E». Если в ходе обучения функция будет находится ниже порогового значения, тогда в нейронной сети (модели) в конечном результате будет состоять из аномалий либо иметь в себе огромное их число.

3.2. Обучение с подкреплением

Основная задача обучения с подкреплением (Reinforcement learning) – это решение задач управления.

В качестве примера программа на основе метода обучения с подкреплением для компьютерной игры (аркада) Breakout, похожей на Pong научилась вести себя иначе. У нее получилось разбивать отверстие в стене так, чтобы пройти на верхнюю часть поля и разбить плиты с другой стороны. Это можно отнести к интеллектуальному поведению.

Другое направление – это управление движением. Например, управлять всеми приводами, одновременно при этом преодолевать препятствия.

И третье направление – это роботы. Описание классической механикой, как именно представить передвижение конечностей робота (например, «лапы» робо-собаки), как устоять и не потерять равновесие и тому подобное.

Система управления (агент), основанные на обучении с подкреплением (reinforcement learning) наблюдает за окружающей средой и получает данные о состоянии этой среды и знаю данные предпринимает определённые действия.

Визуально обучение с подкреплением представлено на рисунке 3.8.

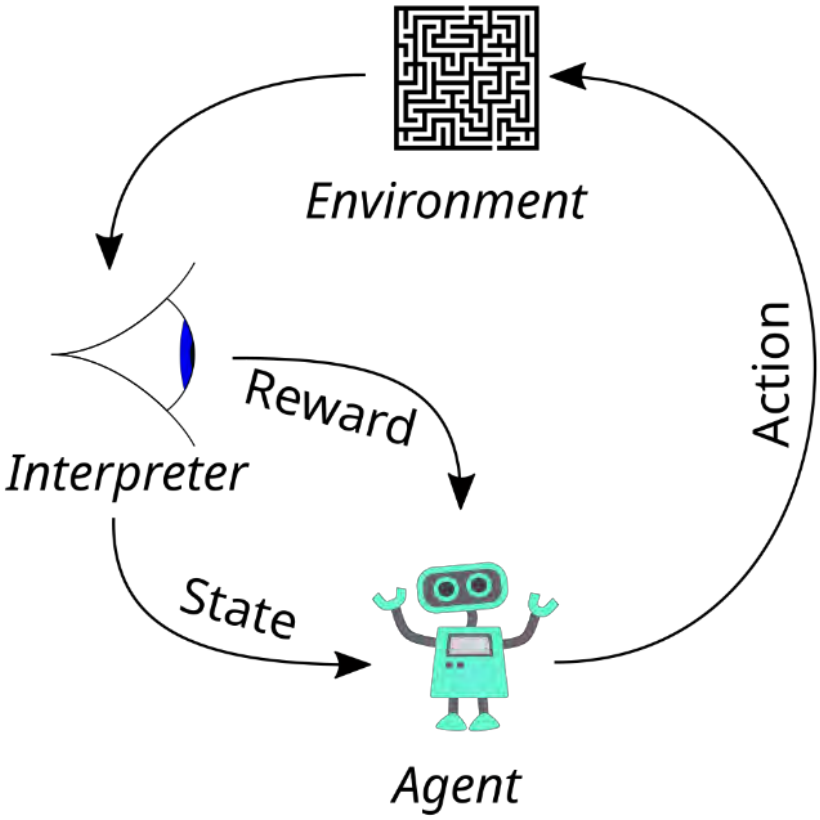


Рисунок 3.8 – Визуальное представление обучения с подкреплением
Environment – Среда; Agent – Система управления «Агент»

Процесс управления или принятия решений: на каждом временном шаге контроллер (агент) получает обратную связь от системы (среды) в виде сигнала состояния и предпринимает ответное действие. Можно предположить, что текущее состояние полностью характеризует состояние системы (марковский процесс принятия решений).

Марковский процесс – это случайный процесс, эволюция которого после любого заданного значения временного параметра не зависит от эволюции, предшествовавшей этому параметру, при условии, что значение процесса в этот момент фиксировано.

Основная проблема заключается в том, что иногда правильные действия системе управления неизвестны.

Основная идея решения данной проблемы дать совет агенту после произошедшего события, предоставляя ему более высокую награду за лучшие действия.

Суть обучения – система управления «Agent» наблюдает за окружающей средой и получает данные состояния этой среды, а также предпринимает действия исходя из полученной информации о среде. При такой системе рассматриваются марковские процессы.

Допущение марковских процессов – состояние системы нынешнее и будущее не зависит от предыстории.

Проблема обучения в системе управления («агента») в том, что правильные действия к выполнению задачи неизвестны. Вариант решения – дать «совет» «агенту» после того, как событие случилось (учитель появляется не на начальном этапе, а когда уже все закончилось): в случае успеха сообщить «агенту» об успехе, в случае неудачи – дать информацию о том, что действие было сделано неверно. Разметка данных происходит после того как завершилось событие. Схематично обучение представлено на рисунке 3.9.

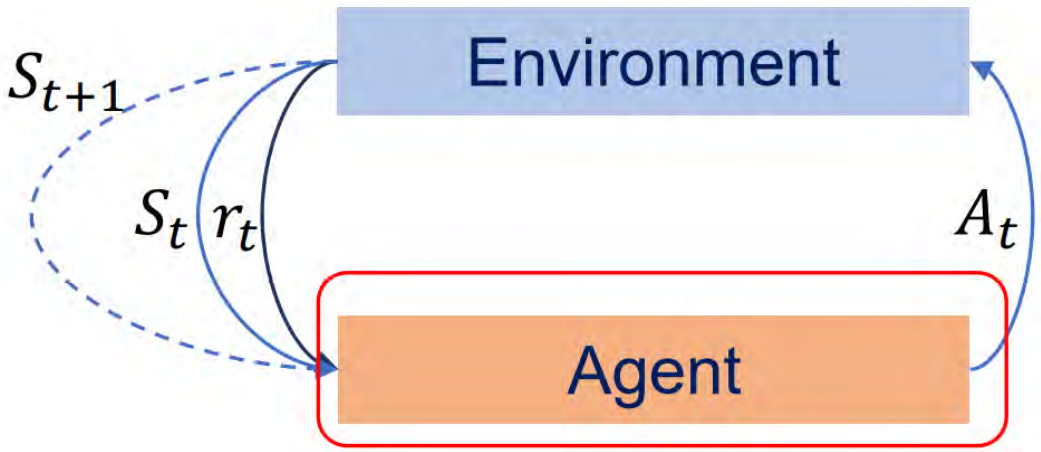


Рисунок 3.9 – Схематичное представление обучения с подкреплением.
Environment – Среда; Agent – Система управления «Агент»

Среда – это то, из чего приходят данные (например, с сенсоров). Среда в которой движется робот – это данные с сенсоров видеоизображения, датчики расстояния до объекта и т.д. Все эти данные приходят из среды.

Агент – это система управления, например, роботом.

Характеристики модели:

– S_t (s_t) – состояние среды (state). Агент в каждый момент времени t получает информацию из среды о состоянии. Определяется матрицей S_t или скалярной величиной s_t;

– A_t (a_t) – действие «агента» (action). Агент в каждый момент времени t предпринимает какие-то действия. Определяется матричной A_t или скалярной a_t величиной;

– r_t – награда (штраф) «агента» (reward). В момент времени t за выполнения действия агент получает награду.

В каждый момент времени «агент» получает информацию среды о состоянии S_t (s_t). И на основании этих данных принимает какое-то действие A_t (a_t). Действие может представлять одно число (повернуть на какой-то угол) или может быть много чисел, если у робота много приводов и на каждый надо выдать действие. И за эти действия система получает награду r_t. Дальше все повторяется – новое состояние, новое действие, новая награда.

В качестве примера рассмотрим задачу о балансировании обратного маятника: необходимо уравновесить шест на подвижной тележке (рис. 3.10).

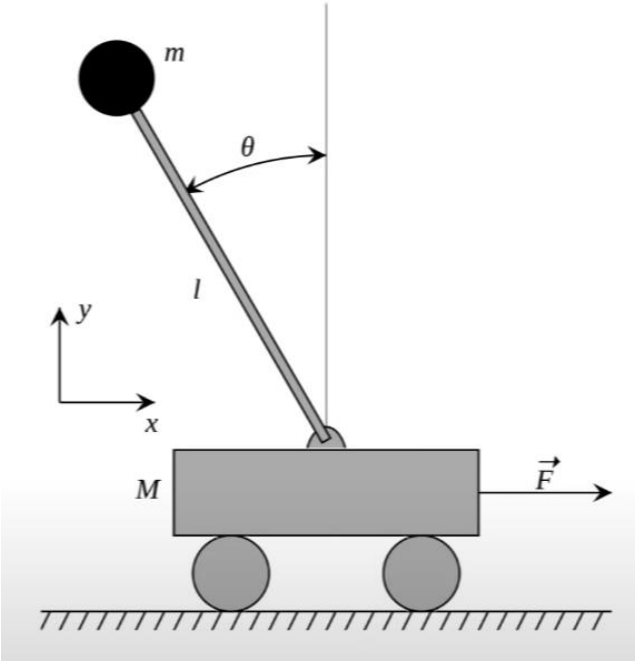


Рисунок 3.10 – Графическое изображение задачи

Состояние объекта характеризуется следующими параметрами: угол, угловая скорость, положение, горизонтальная скорость.

Действие: горизонтальная сила, приложенная к тележке.

Награда: 1 на каждом временном шаге, если шест стоит вертикально.

Суть задачи в том, что маятник в начальный момент времени отклоняется на какой-то угол. Далее необходимо произвести управление тележкой путём приложения силы для выполнения условия: маятник не опрокинулся. Для классических систем управления данная задача является сложной, но применение обучения с подкреплением позволяет найти быстрое решение.

Кроме перечисленных характеристик необходимо рассмотреть ещё одну – будущая награда.

Необходимо оценивать не только ту награду, которую получает «агент» в данный момент времени t, но и предположить, что будет дальше. Данный ряд действий нужно оценить: считается награда не только за первый шаг, но и за последующие шаги. g_t – будущая награда, определяемая по следующей зависимости:

$$g_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^n r_{t+n}\,. \qquad (3.2)$$

где γ – коэффициент уменьшения степени награды (штрафа) при решении задачи в последующем событии (дисконт/discount).

В зависимости от типа системы управления («агента) аппроксимируются следующие функции:

1. $v_t(S_t)$ – оценка будущей награды (штрафа) на основе наблюдения в момент времени t (value function);

2. $q_t(S_t,A_t)$ – оценка будущей награды (штрафа) на основе наблюдения и действия в момент времени t (q- function).

Указанные функции позволяют оценить, что будет дальше (The Critic). Если они делают это правильно, то это очень важная информация.

3. $p_t(S_t)$ – оценка действия на основе наблюдения (policy function).

Policy function – это функция поведения. На основании данных о состоянии в данный момент времени эта функция дает вероятность того, что необходимо выполнить (например, если ракетку нужно двигать вверх при игре в пинпонг, значит лучше так и делать). Эта функция предсказывает действия, которые необходимо совершить (The Actor).

Схематично методы обучения с подкреплением представлены на рисунке 3.11.

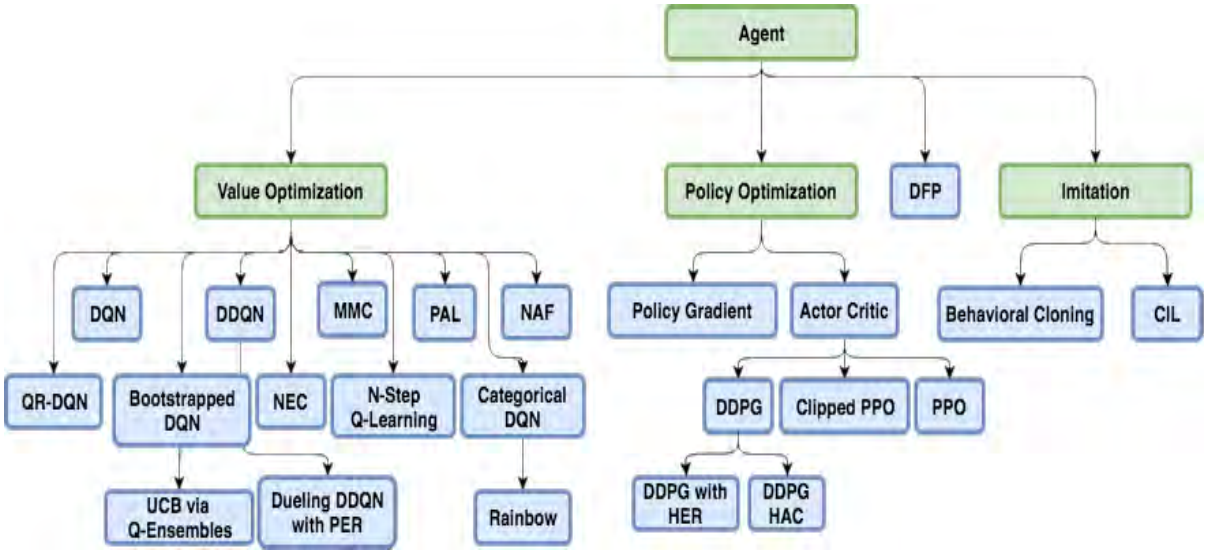


Рисунок 3.11 – Методы обучения с подкреплением

3.3. Оптимизация награды/поведения

3.3.1. Оптимизация поведения

Основная идея метода «Оптимизация поведения» заключается в том, что на основании результатов исследования окружающей среды (environment) обучить модель (critic), которая при данных состояния S_t, и действия A_t предсказывает будущую награду g_t для любого возможного действия на следующем шаге A_{t+1}. Тогда системе управления («агенту») при данных S_t,

A_t из множества возможных дальнейших действий A_{t+1}, следует принимать то, которое приведёт к наибольшей будущей награде g_t.

В каждый момент времени t функция будущей награды будет иметь вид:

$$g_t(S_t, A_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{t^{\triangleright\triangleleft}-1} r_{t^{\triangleright\triangleleft}} \quad (3.3)$$

где γ – коэффициент уменьшения степени награды (штрафа) при решении задачи в последующем событии (дисконт), 0< γ≤1;

$t^{\triangleright\triangleleft}$ – шаг по времени при достижении конечного состояния (terminal state), состояние при котором прекращается процесс. Аргументом в данной функции является состояние и действие в данный момент времени.

Также функцию 3.3 будущей награды можно представить в следующем виде:

$$g_t(S_t, A_t) = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{t^{\triangleright\triangleleft}-2} r_{t^{\triangleright\triangleleft}}) = r_t + r_t q_{t+1}(S_{t+1}, A_{t+1}). \quad (3.4)$$

Тогда наилучшее действие функции будущей награды q_t* в момент времени t описывается уравнением Беллмана:

$$g_t^*(S_t, A_t) = r_t + \gamma \max_A \left[q_{t+1}(S_{t+1}, A_{t+1})\right]. \quad (3.5)$$

Для данного состояния и для каждого действия эта функция $g_t^*(S_t, A_t)$ предсказывает наибольшую возможную награду за все последующие действия.

Процесс наполнения опыта состоит в проигрывании эпизодов. В процессе обучения необходимо достичь минимизации ошибки между обучаемой функцией *Q (S, A)* и оптимальной:

$$Q(S, A) - Q^*(S, A) \Rightarrow \min. \quad (3.6)$$

Тогда в результате обучения система управления будет способна производить действия с максимальной наградой.

В качестве примера стоит рассмотреть задачу «о прогулке по скале». Пример заключается в следующем: необходимо найти кратчайший путь на игровом поле 4×12=48, до цели, не упав с обрыва. Награда за обычный шаг -1, за достижение цели +0, за срыв с обрыва - 100. Коэффициент уменьшения степени награды (дисконт) равен 0,9.

Визуально задача представлена на рисунке 3.12.

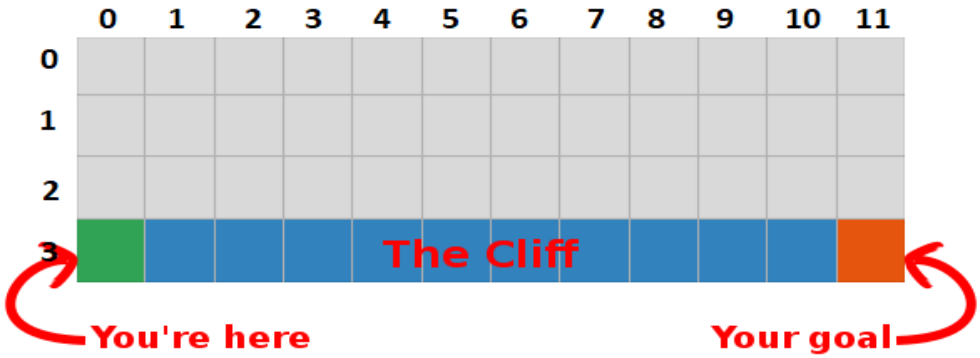


Рисунок 3.12 – Визуальное представление задачи

Требуется найти функцию перемещения Q с использованием параметров состояния S и действия A.

Функция $Q=[q^{(i,j,k)}]$ в определяется зависимости от параметров $S=[s^{(i,k)}]$ и $A=[a^{(k)}]$:

Если «агент» делает много шагов до достижения цели, то он получит маленькую награду. Если «агент» упадет с обрыва, тоже получит за это штраф и в следствие чего получит маленькую награду. Агенту необходимо научиться добираться до точки «terminal state» за минимальное количество шагов.

Первостепенно необходимо определить число возможных состояний. Так как игровое поле в задаче составляет 12 ячеек по горизонтали и 4 ячейки по вертикали, соответственно число возможных состояний будет определяться как произведение числа ячеек по вертикали и горизонтали и будет равно 48 штукам.

Далее из рисунка видно, что имеется ячейка старта и ячейка конца, а также ячейки, отмеченные как «обрыв», при нахождении системы управления на них, штраф будет составлять 100 единиц. За каждый шаг следует штраф в одну единицу. Следовательно, цель задачи – минимизировать число шагов до достижения цели, при условии, что «агент» не будет «наступать» на «обрыв».

Для этого будет представлена таблица-матрица 3.1 возможных перемещений системы управления.

Таблица 3.1

Возможные перемещения системы управления (функция Q)

	$a^{(1)}$ Вверх↑	$a^{(2)}$ Вниз↓	$a^{(2)}$ Вправо→	$a^{(2)}$ Влево←
$s^{(0,0)}$				
$s^{(0,1)}$				
…				
$s^{(2,10)}$	< -1	-1	< -1	< -1
$s^{(2,11)}$	< -1	0	< -1	< -1
…				
$s^{(3,11)}$	0	0	0	0

Алгоритм заполнения всей таблицы Q пока не понятен, но для определения наград в обратном направлении от конечного состояния следует воспользоваться уравнением Беллмана 3.5:

$$g_t^{*(2,11,2)} = 0 + 0{,}9 \max_{a_t+1}[0,0,0,0] = 0,$$

$$g_t^{*(2,10,3)} = -1 + 0{,}9 \max_{a_t+1}[\langle -1,0,\langle -1,\langle -1] = -1,$$

$$g_t^{*(2,9,3)} = -1 + 0{,}9 \max_{a_t+1}[\langle -1,\langle -1,0,\langle -1] = -1{,}9.$$

В конечном итоге направление агента примет следующий вид, показанный на рисунке 3.13.

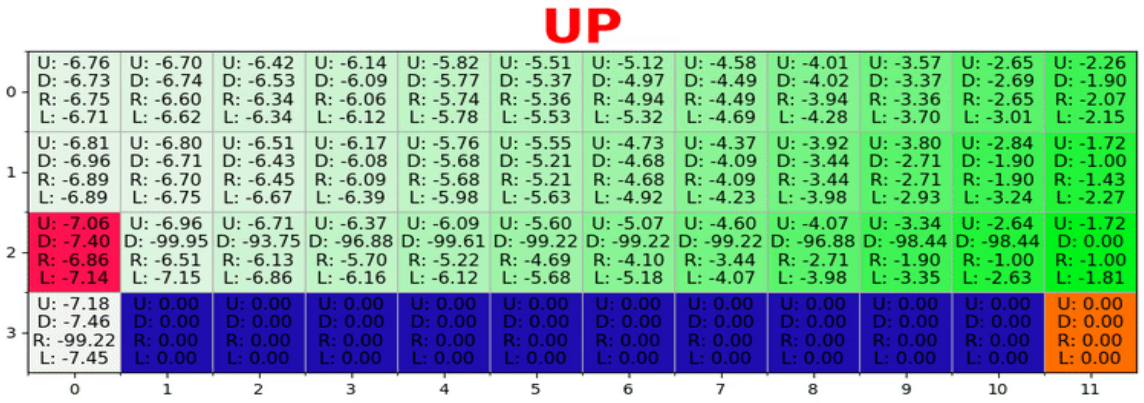


Рисунок 3.13 – Конечная траектория агента,
определённая через уравнения Беллмана

Алгоритм обучения/training algorithm.

Инициализировать функцию *Q(S, A)* случайными значениями или нулями. Задать гиперпараметры: вероятность ε (вероятность случайного или неслучайного действия), скорость обучения α (насколько интенсивно двигается направление градиента).

Для каждого эпизода обучения/

1. Получаются данные о начальном состоянии S_t *(t = 1)* (начальный этап программы, ее старт).

2. Повторять для каждого шага *t* до достижения «terminal state»:

2.1. Для текущего состояния S_t выбрать случайное действие A_t с вероятностью ε, которая может уменьшаться в процессе обучения, иначе действие, для которого значение наибольшее: $A_t = \max_A [Q_t(S_t, A_t)]$.

Элемент случайности добавляется для того, чтобы агент исследовал среду, иначе найдется другой путь, по которому действие будет выполняться всегда одинаково. Такой алгоритм можно назвать «жадным» и степень этой жадности характеризуется вероятностью.

В процессе обучения вероятность ε случайного хода может уменьшаться во времени. Из-за того, что вероятность ε снижается, агент начинает «думать осознанно».

2.2 Далее следует выполнять действие A_t, получить награду r_t и данные о новом состоянии S_{t+1}.

2.3 Если новое состояние S_{t+1} терминальное, то необходимо установить значение целевой функции равной награде $y_t = r_t$. То есть при терминальном состоянии целевая функция y_t приравнивается к полученной награде, так как действие закончено и следующей награды больше нет.

Иначе, целевая функция считается по уравнению Беллмана: $y_t = r_t + \gamma \max_A [q_{t+1}(S_{t+1}, A_{t+1})]$.

2.4 Следующим действием обновляется компонента матрицы Q(S,A): $q(S_t,A_t) = q(S_t,A_t) + \alpha[y_t - q(S_t,A_t)]$

Для каждого элемента предыдущее значение q суммируется произведение коэффициента α на разницу q по уравнению Беллмана и настоящей q.

Если коэффициент α равен 1, то действие q будет считаться по уравнению Беллмана.

Каждое действие циклируется, повторяется до того, как агент обучится. Все, что следует делать агенту, это следовать указаниям.

3.3.2. Оптимизация награды

Основная идея метода «Оптимизация награды» состоит в поиске оптимальной функции вероятности действия (policy function) в каждом данном состоянии $p(A|S, \Theta^{(k)})$, которая максимизирует будущую награду (return):

$$g_t = \sum_{k=t}^{t^{\triangleright\triangleleft}} \gamma^{k-t} r_t. \qquad (3.7)$$

где γ – коэффициент уменьшения степени награды (штрафа) при решении задачи в последующем событии (дисконт), $0 < \gamma \leq 1$; $t^{\bowtie}$ – конечный шаг по времени (terminal state) или бесконечность.

То есть необходимо найти лучшую функцию поведения, которая говорит, что нужно делать при данном состоянии, при данных параметрах нейронной сети с целью максимизации награды.

Пример: задача игры в пинг-понг (рис. 3.13).

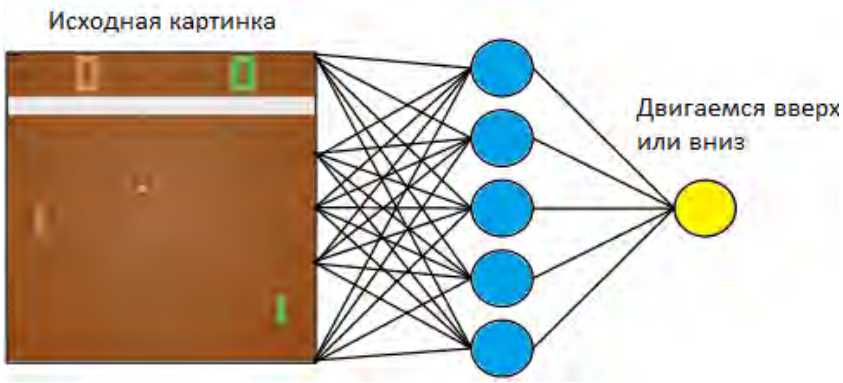


Рисунок 3.13 – Обучение с подкреплением / RL.
Пример программы по управлению ракеткой игры в пинг-понг

State: матрица состояния S_t является картинка размером [210, 160, 3], составленная из значений цветов пикселей цветного изображения состояния игры (точнее разница матриц для двух соседних моментов времени / difference frames).

Action (действие): бинарный выбор движения ракетки a_t = {0.1} вниз или вверх (0 – «DOWN», 1 – «UP»).

Reward (награда): положительная r_t = +1, если соперник пропустил мяч: отрицательная r_t = -1, если агент пропустил.

В данной задаче будет использоваться функция, где выходом будет являться вероятность действия p_i, которая указывает что делать a_i при данном состоянии S_i при данных настройках сети:

$$L(\Theta^{(k)}) = \sum_{i=1}^{m} g_i \ln(p_i(a_i \mid S_i, \Theta^{(k)})) \Rightarrow \max. \tag{3.8}$$

Это повторяется для каждого из движений, это может быть не одна игра, а целый ряд игр.

Алгоритм обучения нейронной сети:

1. Случайным образом назначаются веса $\Theta^{(k)}$ ИНС.

Веса будут $\Theta^{(1)}$, $\Theta^{(2)}$ и будут являться совершенно случайными величинами.

2. Выполняется прогон (rollout) из 100 игровых эпизодов. Все действия выигранных эпизодов считаются правильными и поощряются наградой r_t = +1. Для всех действий проигранных эпизодов награда r_t = -1.

В некоторых исходах программа проигрывает, в некоторый выигрывает. Все действия в выигранных эпизодах поощряются +1, в проигранных – минус 1. А программа стремится максимизировать эту награду.

3. Для каждого действия эпизода рассчитывается значение будущей награды: $g_t = r_{t+j}\gamma^j$ (j=0,1,…).

4. Формируются данные (dataset) прогона: S_i, p_i, g_i. То есть после того, как 100 игр выполнены и имеются данные о состояниях, имеются данные о вероятности действий (сеть предпринимает действия), предпринимались действия (число).Для каждого действия, каждого шага можно рассчитать награду. Эти данные и формируют dataset.

5. Рассчитывается для прогона функция качества:

$$L(\Theta^{(k)}) = \sum_{i=1}^{m} g_i \ln(p_i). \tag{3.9}$$

6. Рассчитывается градиент $\nabla L\left(\Theta^{(k)}\right)$, затем новые значения весов:

$$\theta_{ij}^{(k)} = \theta_{ij}^{(k)} + a\frac{\partial L}{\partial \theta_{ij}^{(k)}}. \tag{3.10}$$

7. Повторение пунктов от 2 до 6 для выполнения неконечных условий.

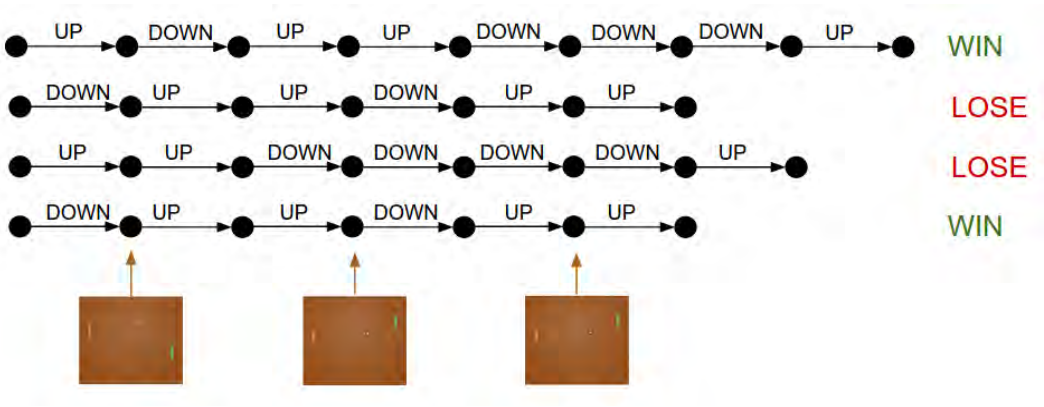


Рисунок 3.14 – Пример работы программы при выполнении 4 игр

Dataset формируется после того как все сыграно. То есть событие уже свершилось и его можно уже оценить. За каждое действия, за каждый шаг назначается награда.

Это диаграмма из 4 игр. Каждый черный кружок – это некоторое состояние игры (три примера состояний показаны внизу), а каждая стрелка – это переход, аннотированный выбранным действием. В данном случае мы выиграли 2 игры и проиграли 2 игры. С помощью Policy Gradient мы возьмем две выигранные игры и слегка подтолкнем каждое действие, которое мы совершили в этом эпизоде. И наоборот, мы также возьмем две проигранные игры и слегка подтолкнем каждое действие, которое мы совершили в этом эпизоде.

Однако если рассматривать процесс в тысячах миллионов игр, то правильное выполнение первого отскока повышает вероятность победы в дальнейшем, так что в среднем вы увидите больше положительных, чем отрицательных изменений. В итоге все будет сделано правильно.

Благодаря предварительной обработке каждый из наших входных данных представляет собой изображение с разницей 80x80 (текущий кадр минус последний кадр). Теперь мы можем взять каждую строку W1, растянуть ее до 80×80 и визуализировать.

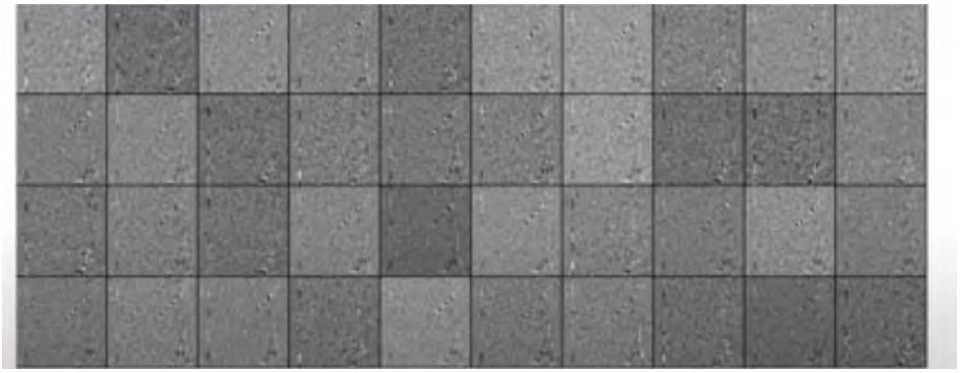


Рисунок 3.15 – Изображения весов 40 нейронов

На рисунке выше изображен набор из 40 (из 200) нейронов в сетке. Белые пиксели – это положительные веса, а черные – отрицательные. Обратите внимание, что некоторые нейроны настроены на определенный след от отскочившего мяча, который кодируется чередованием черного и белого цвета вдоль линии. Мяч может находиться только в одном месте, поэтому эти нейроны работают в режиме многозадачности и будут "стрелять" в несколько мест расположения мяча вдоль этой линии. Чередование черного и белого интересно тем, что по мере движения шарика по трассе активность нейрона будет колебаться в виде синусоиды, а благодаря ReLU он будет «срабатывать» в отдельных и дискретных позициях неконечных условий.

4. ГЛУБОКОЕ ОБУЧЕНИЕ И ЕГО ПРИЛОЖЕНИЯ

4.1. Глубокие сверточные нейронные сети

Если использовать нейронные сети прямого распространения, возникает проблема обработки больших изображений. Например: если на вход нейронной сети подается изображение размером 1000 на 1000 пикселей, то количество входных нейронов составит 10^6 (миллион). Затем, если есть следующий слой, например, содержащий 10^2 (сто) нейронов, то количество весовых параметров, описывающих эти два слоя, уже составит сто миллионов. Это значительное количество, и обучение таких сетей становится сложной задачей. Проблемы точности могут также возникнуть при работе с изображениями (рис. 4.1).

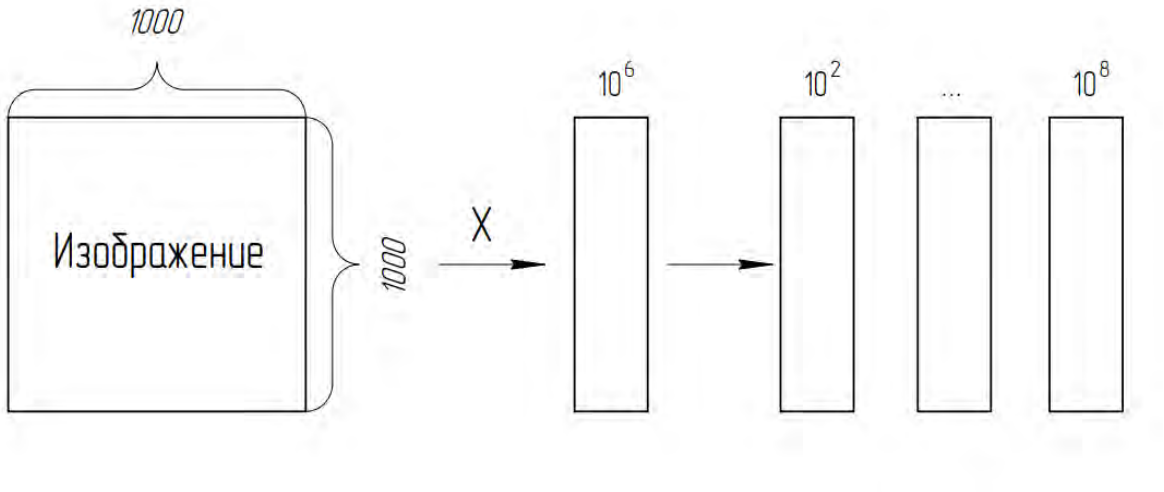


Рисунок 4.1 – Нейронная сеть прямого распространения

Для преодоления эти ограничений, необходимо применять новые архитектуры нейронных сетей. Одно из решений в 1950-х годах было экспериментально продемонстрировано на примере обработки изображений, начиная с исследований на кошках. Суть эксперимента заключалась в следующем: электроды подключаются к мозгу кошки, к конкретным нейронам и сигнал с электрода передавался на аудиомонитор. Задача состояла в том, чтобы определить, на какие изображения реагирует нейрон. Во время эксперимента замечено, что некоторые нейроны реагируют на линии изображения.

Это открытие носит случайный характер: когда изображение пятна показывали кошке, нейрон не реагировал. Однако, когда изображение перемещалось и на нем появлялась тень, некоторые нейроны начинали реагировать и подавать сигнал на аудиомонитор. В конечном итоге обнаружено, что нейроны реагируют на простые признаки, такие как линии. Сигнал, в зависимости от типа линии, имел разный уровень.

Свёрточная нейронная сеть (СНС) использует особенности зрительной части коры головного мозга, в которой простые клетки активируются на простые признаки, а сложные на комбинацию активаций простых.

СНС связана с математической операцией свертки для понижения размеров матриц. Свёрточные нейронные сети обычно являются глубокими.

В 1998 году представлена одна из первых известных архитектур нейронных сетей, известная как «линейная нейронная регрессия» (ЛНР). Однако настоящий прорыв в области искусственного интеллекта произошел в 2012 году, когда участники конкурса ImageNet смогли разработать модели, способные распознавать порядка миллиона изображений и классифицировать их на 1000 классов.

До этого момента точность классификации была довольно низкой, с ошибкой около 30 процентов. Однако в 2012 году эта ошибка была существенно снижена до 16 процентов. Этот значительный прогресс достигнут благодаря применению глубоких сверточных нейронных сетей, которые долго обучали.

Эти сети включают в себя операции свертки, операции выбора максимального значения, а также функции активации, такие как функция ReLU (Rectified Linear Unit), которая обнуляет отрицательные сигналы и передает положительные без изменений. Такие операции повторяются много раз внутри сети.

В конце схемы нейронной сети обычно находится несколько полносвязанных слоев, которые объединяют все предыдущие вычисления. Выходной слой содержит количество нейронов, равное количеству классов в задаче классификации, и обычно используется функция «softmax» для преобразования выходных значений в вероятности принадлежности к каждому классу (рис. 4.2).

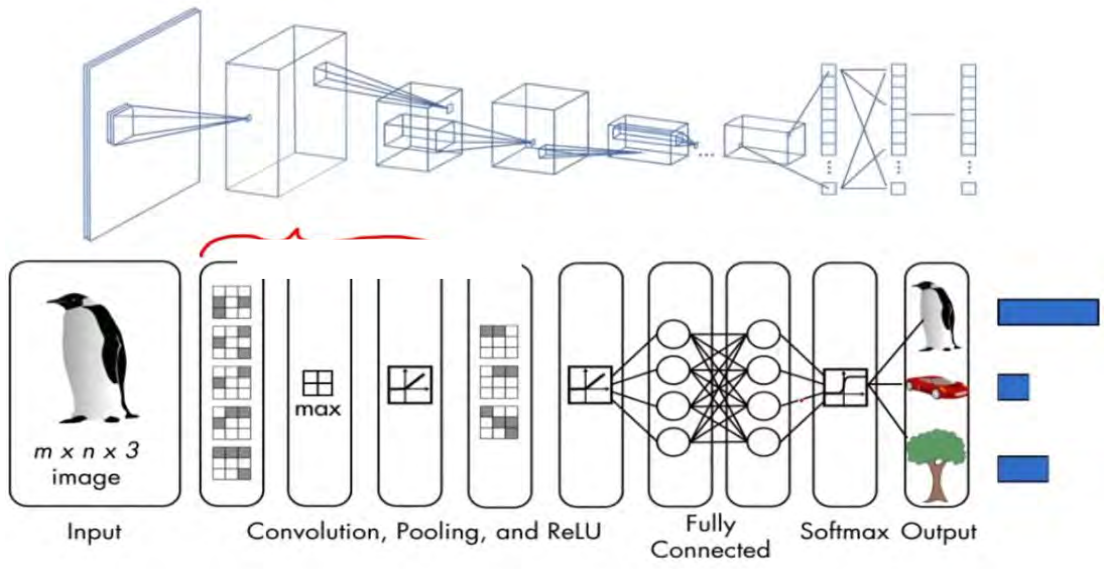


Рисунок 4.2 – Схема работы сверточных нейронных сетей

На вход подается изображение и сеть выдает прогноз, к какому классу оно относится с наибольшей вероятностью. Суть заключается в том, что нейросеть автоматически выявляет признаки на изображении без необходимости вручную задавать их (например, распознавать глаза, нос и т.д.) – это вершина области искусственного интеллекта глубокого обучения. Характерной чертой глубокого обучения является автоматическое выявление признаков.

Если посмотреть на сверточные нейронные сети, можно увидеть так называемые «карты признаков» (feature maps). Первые слои сети выделяют простые признаки, а последующие слои выделяют все более сложные. В результате происходит классификация изображения с высокой точностью (рис. 4.3).

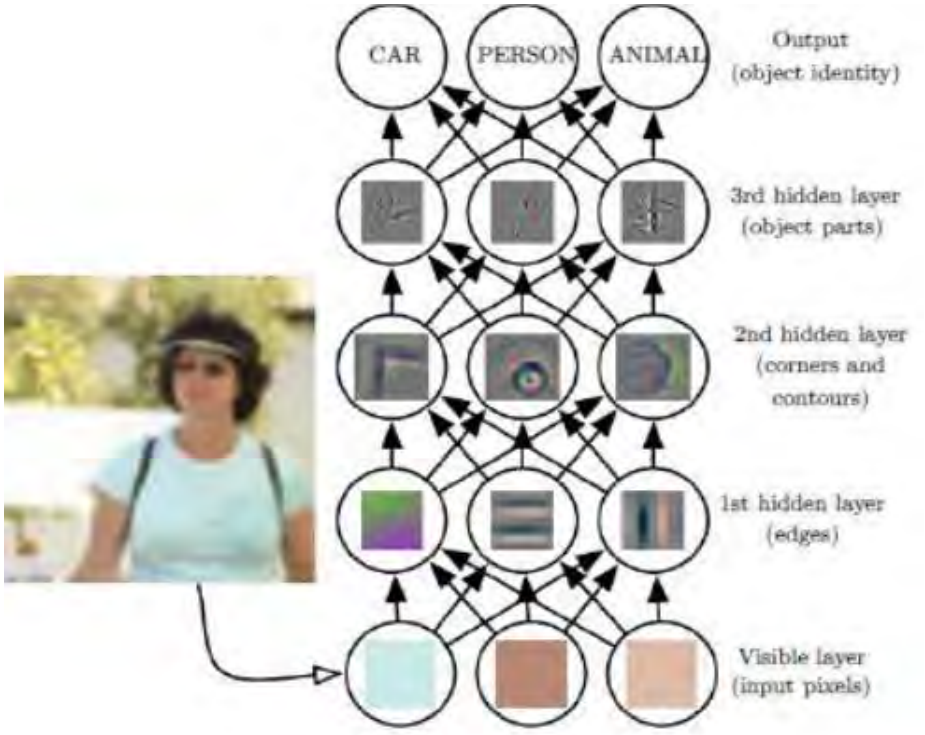


Рисунок 4.3 – Схема обработки изображения

Основная операция в сверточных нейронных сетях – это свертка, математическая операция, которая применяется к каждому пикселю изображения. Изображение представляется в виде матрицы, где каждый пиксель обладает определенной яркостью. Далее к этой матрице применяются ядра, которые помогают выделять различные признаки на изображении.

Ядра применяются в сверточных нейронных сетях путем сканирования изображения с помощью окна, которое двигается по изображению. Сначала выбирается фрагмент изображения размером, например, 3 на 3 пикселя. Этот фрагмент можно обозначить как «x^*», что означает кусок изображения.

Операция в тензорном анализе, которая выполняется над этими кусками изображения, называется операцией скалярного произведения, или сверт-

кой. Это по существу поэлементное умножение двух матриц. На каждом шаге каждый элемент матрицы «*x*», который представляет фрагмент изображения, умножается на соответствующий элемент матрицы «*t*». Размер ядра определяет количество таких операций, которые выполняются, обычно от одного до трех для ядер размером 3×3 (рис. 4.4).

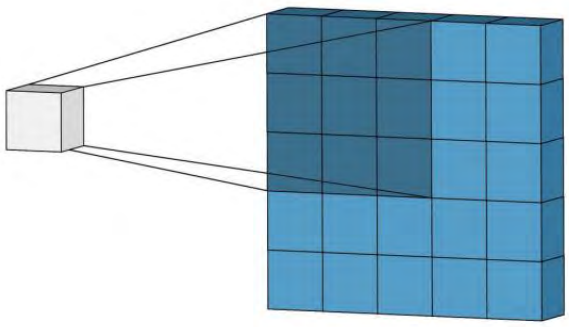


Рисунок 4.4 – Сканирование изображения и разбиение на зоны

Эту операцию можно представить в виде одной операции, если предварительно подготовить изображение и ядро. Для этого значения пикселей, попадающих в окно, могут быть преобразованы в одну строку. Таким образом, каждое окно изображения превращается в строку чисел. То же самое делается и для ядра, которое становится столбцом чисел (рис. 4.5).

$$X = \begin{pmatrix} \begin{matrix} 3 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 2 & 2 & 3 \\ 2 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 1 \end{matrix} \end{pmatrix} \rightarrow A^{(1)} = \begin{pmatrix} \begin{matrix} 3 & 3 & 2 & 0 & 0 & 1 & 3 & 1 & 2 \\ 3 & 2 & 1 & 0 & 1 & 3 & 1 & 2 & 2 \\ 2 & 1 & 0 & 1 & 3 & 1 & 2 & 2 & 3 \\ 0 & 0 & 1 & 3 & 1 & 2 & 2 & 0 & 0 \\ 0 & 1 & 3 & 1 & 2 & 2 & 0 & 0 & 2 \\ 1 & 3 & 1 & 2 & 2 & 3 & 0 & 2 & 2 \\ 3 & 1 & 2 & 2 & 0 & 0 & 2 & 0 & 0 \\ 1 & 2 & 2 & 0 & 0 & 2 & 0 & 0 & 0 \\ 2 & 2 & 3 & 0 & 2 & 2 & 0 & 0 & 1 \end{matrix} \end{pmatrix};$$

Рисунок 4.5 – Сканирование матрицы (input)

После выполнения операции свертки, полученную матрицу можно преобразовать обратно в квадратную форму. Таким образом, процесс свертки преобразует входные данные в матрицу, содержащую новые значения, которые представляют результаты операции.

Матрица ядра, свертка и развертка в матрицу 3×3 описывается выражениями 4.1, 4.2.

$$\theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix} \end{pmatrix} \rightarrow \theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 2 \\ 2 \\ 0 \\ 0 \\ 1 \\ 2 \end{pmatrix} \end{pmatrix}. \quad (4.1)$$

$$Z^{(2)} = A^{(1)}\theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 12 \\ 12 \\ 17 \\ 10 \\ 17 \\ 19 \\ 9 \\ 6 \\ 14 \end{pmatrix} \end{pmatrix} \rightarrow Z^{(2)} = \begin{pmatrix} \begin{pmatrix} 12 & 12 & 17 \\ 10 & 17 & 19 \\ 9 & 6 & 14 \end{pmatrix} \end{pmatrix}. \quad (4.2)$$

Ядро представляет собой матрицу с определенными элементами, которые определяются при его создании. После применения этого ядра к изображению получается новое изображение, где выделяются определенные характеристики в зависимости от содержания ядра.

Например, если применить ядро, которое выделяет горизонтальные линии, то на полученном изображении будут выделены горизонтальные структуры или объекты, близкие к горизонтальным. Аналогично, применение транспонированного ядра, выделит вертикальные линии на изображении (рис. 4.6).

Рисунок 4.6 – Пример применения ядра, которое выделяет вертикальные линии

Компоненты этих ядер обычно не задаются заранее, а определяются в процессе обучения сверточной нейронной сети. Их значения сначала инициализируются случайным образом, а затем корректируются в процессе обучения для минимизации ошибки.

Операцию свёртки обычно комбинируют с операциями дополнения, группирования и шагания.

У операции дополнения есть атрибуты, такие как дополнение нулями. Этот атрибут добавляет нулевые элементы к изображению перед операцией свертки. Это часто используется для того, чтобы обеспечить корректную обработку краев изображения (рис. 4.7).

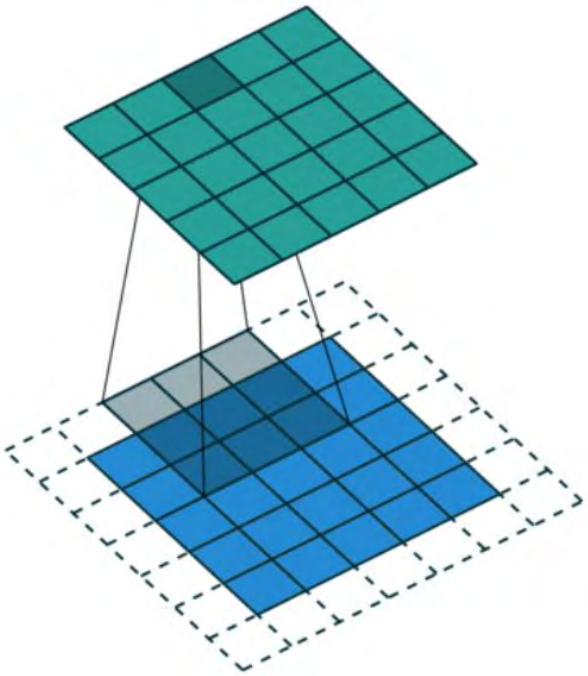


Рисунок 4.7 – Дополнение изображения «нулями»

Свертка может применяться с различными вариациями в зависимости от задачи.

Во-первых, добавление дополнительных границ с нулями (padding) позволяет фильтру проработать всё изображение. Если это не сделать, то только краевая часть фильтра коснется краев изображения.

Во-вторых, результат свертки может иметь такой же размер, как и исходное изображение. Это удобно, когда требуется сохранить пространственную информацию изображения.

С другой стороны, чтобы экономить количество параметров и ускорить обучение, можно использовать атрибут «шагания» (striding). Шаг определяет, через сколько пикселей применяется ядро. Например, шаг 2 означает, что ядро применяется, перескакивая через каждый второй пиксель, что приводит к уменьшению размера результирующей матрицы.

После свертки может быть выполнена операция «групирование» (pooling). Например, операция max-pooling разбивает изображение на области размером 2×2. Суть операции заключается в том, что произвести выбор максимальное значение из каждой области. Это помогает уменьшить размер матрицы и количество параметров.

Применения свертки можно рассмотреть на конкретном примере. Предположим, есть изображение цифры «4» размером 3×3, представленное в градациях от нуля до единицы. Представим данное изображение x сканирование изображения цифры «4», уже дополненное нулями (рис. 4.8).

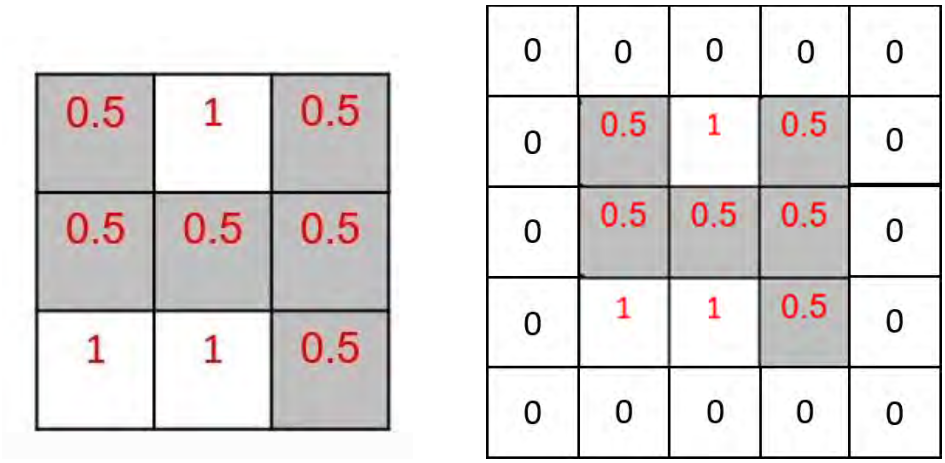


Рисунок 4.8 – Дополнение изображения

Применяем к нему ядро с весами, представленными в матрице, и получаем результат свертки, в виде матрицы 4.3.

$$X = \begin{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}5 & 1 & 0{,}5 & 0 \\ 0 & 0{,}5 & 0{,}5 & 0{,}5 & 0 \\ 0 & 1 & 1 & 0{,}5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{pmatrix} \rightarrow A = \begin{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0{,}5 & 1 & 0 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 0{,}5 & 1 & 0{,}5 & 0{,}5 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 1 & 0{,}5 & 0 & 0{,}5 & 0{,}5 & 0 \\ 0 & 0{,}5 & 1 & 0 & 0{,}5 & 0{,}5 & 0 & 1 & 1 \\ 0{,}5 & 1 & 0{,}5 & 0{,}5 & 0{,}5 & 0{,}5 & 1 & 1 & 0{,}5 \\ 1 & 0{,}5 & 0 & 0{,}5 & 0{,}5 & 0 & 1 & 0{,}5 & 0 \\ 0 & 0{,}5 & 0{,}5 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0{,}5 & 0{,}5 & 0{,}5 & 1 & 1 & 0{,}5 & 0 & 0 & 0 \\ 0{,}5 & 0{,}5 & 0 & 1 & 0{,}5 & 0 & 0 & 0 & 0 \end{pmatrix} \end{pmatrix} \quad (4.3)$$

По стандартной процедуре, производится выбор элементов из матрицы 3×3 и производится разворачивание их в ряд, проходя по всем возможным окончаниям с шагом 1. Далее берётся фильтр, или ядро, по стандартной процедуре, и также производится развертывание его в столбец.

Тогда процедура свертки сводится к матричному умножению, где x представляет собой входные данные, а ядро θ играет роль весов:

$$\theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{pmatrix} \rightarrow \theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 2 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \end{pmatrix}; Z^{(2)} = A^{(1)}\theta^{(1)} = \begin{pmatrix} \begin{pmatrix} 2{,}5 \\ 3{,}5 \\ 2{,}5 \\ 3 \\ 4 \\ 2{,}5 \\ 3{,}5 \\ 4 \\ 2{,}5 \end{pmatrix} \end{pmatrix}; Z^{(2)} = \begin{pmatrix} \begin{pmatrix} 2{,}5 & 3 & 3{,}5 \\ 3{,}5 & 4 & 4 \\ 2{,}5 & 2{,}5 & 2{,}5 \end{pmatrix} \end{pmatrix}. \quad (4.4)$$

После применения функции активации матрицы $Z^{(2)}$.

Результат можно преобразовать и быть представлен в виде квадратной матрицы (рис. 4.9).

Рисунок 4.9 – Результат свертки изображения цифры «4»

Процесс свертки состоит в том, чтобы пройти ядром по изображению, умножая каждый элемент изображения на соответствующий элемент ядра, а затем суммируя результаты. Это позволяет нам выделить горизонтальные грани на изображении.

При обработке цветного изображения используются три канал: красный R, зелёный G, синий B. При свертке в цветном изображении стоит отметить, что она происходит отдельно для каждого канала RGB. Совокупность трёх ядер образует фильтр. Это означает, что для каждого канала применяются свои ядра и результаты свертки для каждого канала складываются вместе и получается одна матрица. Дополнительно к матрице добавляется смещение (bias) какое-то число, аналогично параметру линейной регрессии $h=\theta_0+ \theta_1 x$ (рис. 4.10).

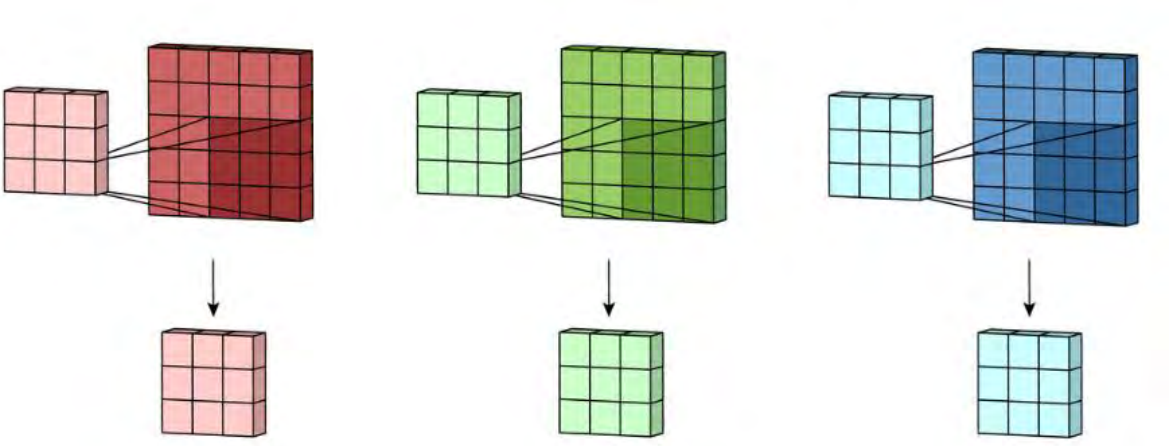


Рисунок 4.10 – схема свертки цветного изображения

Происходит объединение основных цветов изображения после свертки и добавления к ним определенного фильтра, влияющий на контрастность, яркость изображения (рис. 4.11).

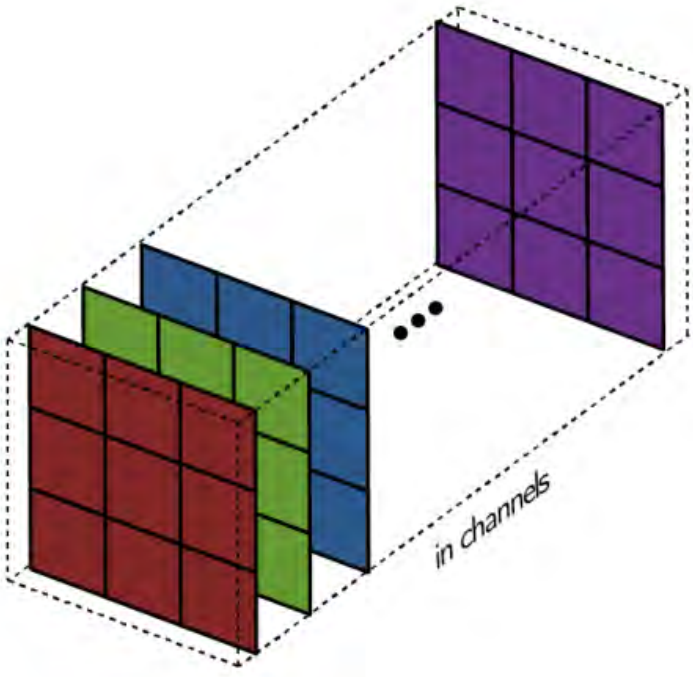


Рисунок 4.11 – результат свертки цветного изображения

Например, если у нас есть изображение размером 5 на 5 пикселей и три канала (RGB), и мы применяем два фильтра, каждый размером 3 на 3 пикселя, то для каждого канала будет выполнена свертка с соответствующим фильтром. Затем результаты сверток складываются, и к ним добавляется смещение (рис. 4.12).

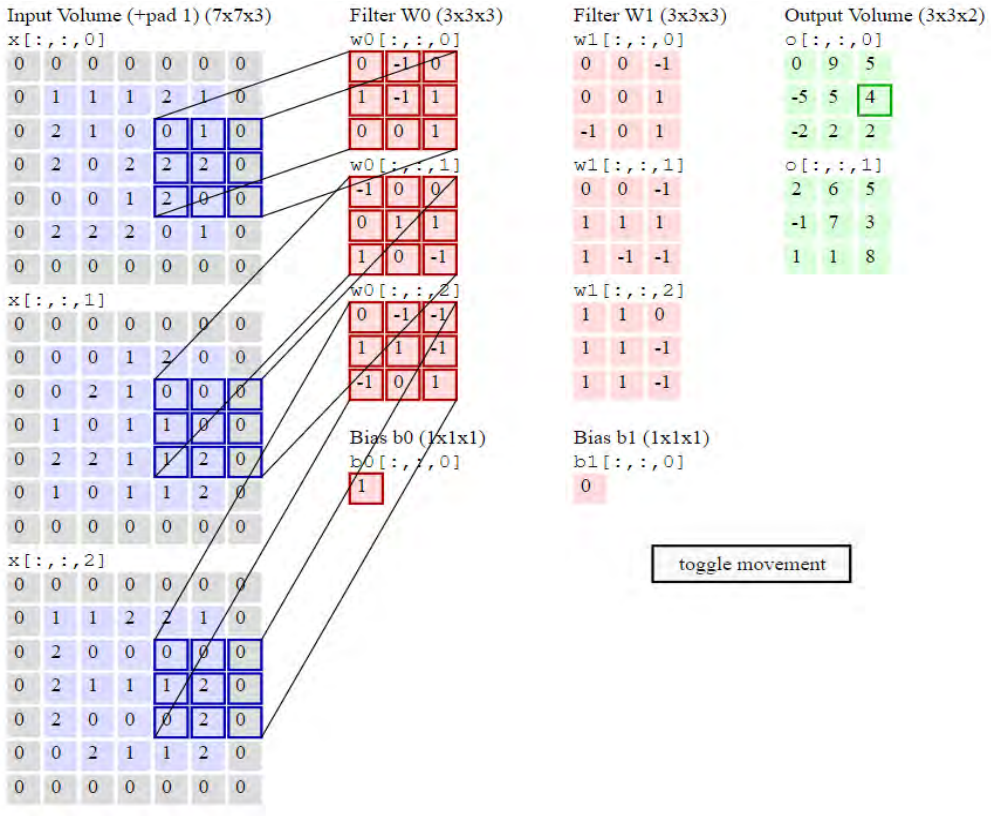


Рисунок 4.12 – Пример матричного свертывания изображения

Рассмотрим конкретный пример применения свертки к цветному изображению дома. В этом случае используется два фильтра для выделения различных признаков, например, крыши и окон. Каждый фильтр имеет свои веса, и они применяются к каждому каналу изображения (красному, зеленому и синему) по отдельности. Результаты сверток для каждого канала объединяются, и к ним добавляется смещение. Это позволяет модели выделять интересующие признаки на изображении, а функция активации применяется к полученным результатам для создания финальных признаков («фичей»), которые затем передаются в следующие слои нейронной сети для дальнейшей обработки.

Для каждого фрагмента изображения размером 3×3 будет выполнена операция свертки с фильтром Hydra. Количество таких операций будет равно количеству фрагментов размером 3×3 на изображении. Например, если изображение имеет размер 5×5 пикселей, то можно провести 9 операций свертки (так как количество возможных фрагментов размером 3 на 3 на изображении равно 9).

Результатом каждой операции свертки будет матрица размером 3 на 3. Это можно вычислить заранее, необходимо только учесть размер фильтра и шаг, с которым он применяется к изображению. Таким образом, результатом операции свертки будет матрица, размер которой будет равен размеру фильтра, а количество таких операций будет зависеть от размера изображения и шага свертки.

Размером матрицы активации первого слоя получается 9 на 27 из-за того, что каждое окно свертки размером 3 на 3 превращается в строку из 9 элементов, и таких окон будет 9 на каждом канале изображения, в данном случае 3 канала. Таким образом, получается 27 столбцов.

Далее, если применяются два фильтра размером 3 на 3 на 3, то каждый из них имеет 27 компонентов (поскольку каждый фильтр представляется в виде столбца). Так как есть два фильтра, общее количество компонентов для объединенной матрицы фильтров будет 27 × 2 = 54.

Зная размеры этих матриц, можно вычислить размер промежуточного значения *z* по стандартной процедуре умножения матриц: если матрица активации первого слоя имеет размерность 9 на 27, а объединенная матрица фильтров имеет размерность 27 на 2, то размерность промежуточного значения *z* будет 9 на 2.

Так, произведение матриц Θ будет иметь размерность 9 на 2, потому что в соответствии с правилами умножения матриц, размеры, которые не касаются в процессе умножения, сокращаются. В результате получается выходная матрица размером 9 на 2.

Каждому элементу этой матрицы *z* будет прибавляться значение bias. В матричной форме это выглядит так: матрица имеет первый столбец с 9 элементами, равными 1, и второй столбец с 9 элементами, равными 0 (это значения bias). Таким образом, представляет собой эти фильтры.

Если к этим значениям применить функцию активации, то получится две карты признаков: карта признаков 1 и карта признаков 2. Эти карты признаков можно воспринимать как изображения, которые показывают, что нейронная сеть «видит» в данных. В архитектуре AlexNet количество таких фильтров довольно велико. Например, во втором слое есть 96 фильтров. В результате свертки и применения функции активации получается 96 карт признаков, которые выделяют различные характеристики в изображении, такие как горизонтальные и вертикальные линии, наклонные линии и т.д.

Затем применяется операция «макс-пулинга», которая уменьшает размер этих карт признаков. Этот процесс повторяется, применяя свертку и «макс-пулинг,» и количество фильтров обычно увеличивается с увеличением глубины сети.

Поиск наилучшей функциональной зависимости множества *Y* от множества *X* по критерию минимума некоторой функции качества с помощью свёрточных нейронных сетей, с количеством слоев *l*, количество нейронов *k*-ом слое n_k (n_i – количество классов), параметризованной весами $\Theta^{(k)}$, на основании анализа *m* пар значений *Y, X*.

Функция качества для сверточных нейронных сетей выглядит следующим образом:

$$J(\theta^{(k)}) = -\frac{1}{m}\sum_{i=0}^{m}\sum_{j=1}^{n_1}(y_j^{(i)}\ln(h_j^{(i)}) + (1-y_j^{(i)})(\ln(1-h_j^{(i)})) \Rightarrow \min \qquad (4.5)$$

Алгоритм обучения сверточных нейронных сетей.

1. Задать начальные значения компонент матрицы $\Theta^{(k)}$ случайным образом.

2. Рассчитать вектор градиента $\nabla J = [\![\partial J / \partial \theta_{ij}^{(k)}]\!]$ методом обратного распространения ошибки.

3. Найти новые значения компонент Θ: $\theta_{ij}^{(k)^H} = \theta_{ij}^{(k)^C} - \alpha \frac{\partial J}{\partial \theta_{ij}^{(k)}}$.

4. Повторять пп. 2–3 до достижения минимума J: $J^H - J^C < \delta$ или количество итераций достигнет максимального значения: $N_{\text{итерации}} > N_{max}$.

5. Вывод результатов: результатом является все матрицы $\Theta^{(k)}$.

4.2. Настройка обучения сверточных нейронных сетей

Настройка обучения в контексте глубокого обучения, включая сверточные нейронные сети (CNN Training settings), представляет собой важный этап, определяющий эффективность и производительность модели.

Существует несколько ключевых аспектов настройки обучения, которые влияют на результаты обучения.

1. *Скорость обучения* α (learning rate) – это один из наиболее критических параметров обучения нейронных сетей. Она определяет величину шага, с которым корректируются веса модели в процессе градиентного спуска.

Постоянная скорость обучения. Данный метод прост и понятен, где скорость обучения остается неизменной в течение всего процесса обучения. Он подходит для стабильных задач и датасетов, но может быть неэффективным в случае сложных данных.

Адаптивная скорость обучения. Данный подход регулирует скорость обучения в зависимости от характеристик обучения. Методы, такие как AdaGrad, RMSProp и Adam, автоматически адаптируют скорость обучения на основе градиентов и истории обновлений, обеспечивая более эффективное обучение.

Расписание скорости обучения. Этот метод изменяет скорость обучения с течением времени в соответствии с заранее заданным расписанием. Например, скорость обучения может уменьшаться экспоненциально с каждой эпохой.

2. *Погрешность* δ *и количество итераций* N_{max}. Погрешность (или ошибка) определяется как разница между прогнозируемыми и фактическими значениями на валидационном или тестовом наборе данных. Минимизация этой погрешности является целью обучения.

Мониторинг погрешности. Регулярное отслеживание производительности модели на валидационном наборе данных помогает оценить ее способность к обобщению на новые данные и предотвратить переобучение или недообучение.

Графики обучения. Визуализация функции потерь на обучающем и валидационном наборах данных позволяет анализировать процесс обучения и выявлять проблемы, такие как переобучение.

Количество итераций (эпох). Этот параметр определяет, сколько раз весь тренировочный набор данных проходит через модель. Контроль за количеством итераций может быть осуществлен методом early stopping, который прекращает обучение при отсутствии улучшения производительности.

Объемный и точный анализ этих аспектов настройки обучения играет решающую роль в достижении оптимальных результатов глубокого обучения, особенно в случае сверточных нейронных сетей.

3. *Количество данных для градиентного метода.*

Размер тренировочного набора данных. Большой тренировочный набор данных обычно позволяет модели лучше обучаться и лучше обобщать на новые данные. Это помогает предотвратить переобучение и улучшить общую производительность модели.

Однако важно также учитывать, что большие объемы данных могут потребовать больших вычислительных ресурсов и времени на обучение.

Разбиение данных. Важно разбить доступный dataset на тренировочный, валидационный и тестовый наборы данных. Обычно рекомендуется использовать около 60-80 % данных для тренировочного набора, 10-20 % для валидационного и 10-20 % для тестового.

Этот баланс позволяет обучать модель на достаточном объеме данных, валидировать ее производительность и проверить ее обобщающую способность на новых данных.

Учет сложности задачи. Сложные задачи могут потребовать больших объемов данных для эффективного обучения модели. Например, для задачи классификации изображений на большое количество классов может потребоваться большое количество обучающих данных для обучения модели с высокой точностью.

В некоторых случаях использование предварительно обученных моделей (transfer learning) может помочь решить проблему нехватки данных, позволяя использовать заранее обученные параметры модели на подобных задачах.

4. Регуляризация (regularization) – это методы контроля за переобучением модели путем добавления дополнительных ограничений к функции потерь. Это может включать в себя *L1* и *L2* регуляризацию, отсев нейронов (dropout), а также более сложные методы, такие как аугментация данных.

Подбор оптимальных параметров регуляризации может помочь улучшить обобщающую способность модели и снизить вероятность переобучения.

5. Аугментация данных. Для увеличения разнообразия обучающего набора данных и предотвращения переобучения можно использовать методы аугментации данных. Это может включать в себя случайные преобразования изображений (например, повороты, отражения, изменения масштаба) или другие методы изменения данных.

6. «*Дропаут»* (от англ. dropout) – метод регуляризации искусственных нейронных сетей, предназначенный для уменьшения переобучения сети за счёт предотвращения сложных коадаптаций отдельных нейронов на тренировочных данных во время обучения.

Принцип метода «дропаут» состоит в том, что во время обучения нейронной сети производится случайное исключение определённого количества случайных нейронов (например 30%), которые могут находиться как в скрытых, так и видимых слоях, на разных итерациях (эпохах). Результатом применения данного метода является то, что в сети больший вес получают более обученные.

Использование метода «дропаут» позволяет значительно увеличить скорость обучения нейронной сети, качество обучения на тренировочных данных. Также повышает качество предсказаний модели на новых тестовых данных.

Основным недостатков метода «дропаут» является увеличение время обучения сети dropout обычно в 2–3 раза, чем у стандартной нейросети с такой же архитектурой.

7. Трансферное обучение – это метод обучения нейронных сетей, при котором знания, полученные в результате обучения на одной задаче или наборе данных, применяются к другой связанной задаче или наборе данных. В контексте сверточных нейронных сетей (CNN), трансферное обучение часто используется для передачи знаний от предварительно обученных моделей на больших наборах данных к новым задачам с меньшим количеством данных.

4.3. Примеры архитектур и приложений глубокого обучения сетей

Основным критерием классифицирования нейронных сетей является её структура. В каждой нейронной сети имеет первый слой нейронов, который называется входным. Данный слой не выполняет вычислений и преобразований, а производит приём и распределение сигналов по остальным нейронам. Данный слой является единственным, который присутствует во всех типах нейронных сетей.

Принцип работы нейронных сетей включает в себя нижеперечисленные этапы.

Предварительное обучение модели. Сначала модель CNN обучается на большом наборе данных, часто содержащем миллионы изображений, для решения связанных задач, например, классификации изображений в ImageNet.

Использование предварительно обученных признаков. Затем эта предварительно обученная модель используется для извлечения признаков из изображений нового набора данных, на котором требуется решить другую задачу, например, классификацию изображений в более узком домене.

Тонкая настройка (Fine-tuning). После извлечения признаков из предварительно обученной модели, эти признаки могут быть использованы для обучения новой модели или тонкой настройки предварительно обученной модели на новом наборе данных. Это позволяет модели лучше адаптироваться к специфическим характеристикам новой задачи или домена.

Преимущества:

– использование предварительно обученных признаков. Предварительно обученные модели содержат знания, полученные из больших наборов данных, что обеспечивает хорошее извлечение признаков для новых задач;

– улучшение производительности. Трансферное обучение позволяет улучшить производительность моделей на небольших наборах данных, которые могут быть недостаточными для обучения с нуля;

– экономия времени и ресурсов. Поскольку предварительно обученные модели уже содержат веса и знания, извлеченные из больших наборов данных, это позволяет сократить время и вычислительные ресурсы, требуемые для обучения модели на новых задачах.

Ограничения:

– Переносимость знаний. Предварительно обученные модели могут не всегда быть хорошо адаптированы к новой задаче или домену, и требуют тонкой настройки и/или дополнительных модификаций;

– переобучение. Существует риск переобучения при тонкой настройке предварительно обученных моделей на небольших наборах данных, особенно если количество данных недостаточно для эффективного обучения.

Обобщённая классификация нейросетей представлена на рисунке 4.13.

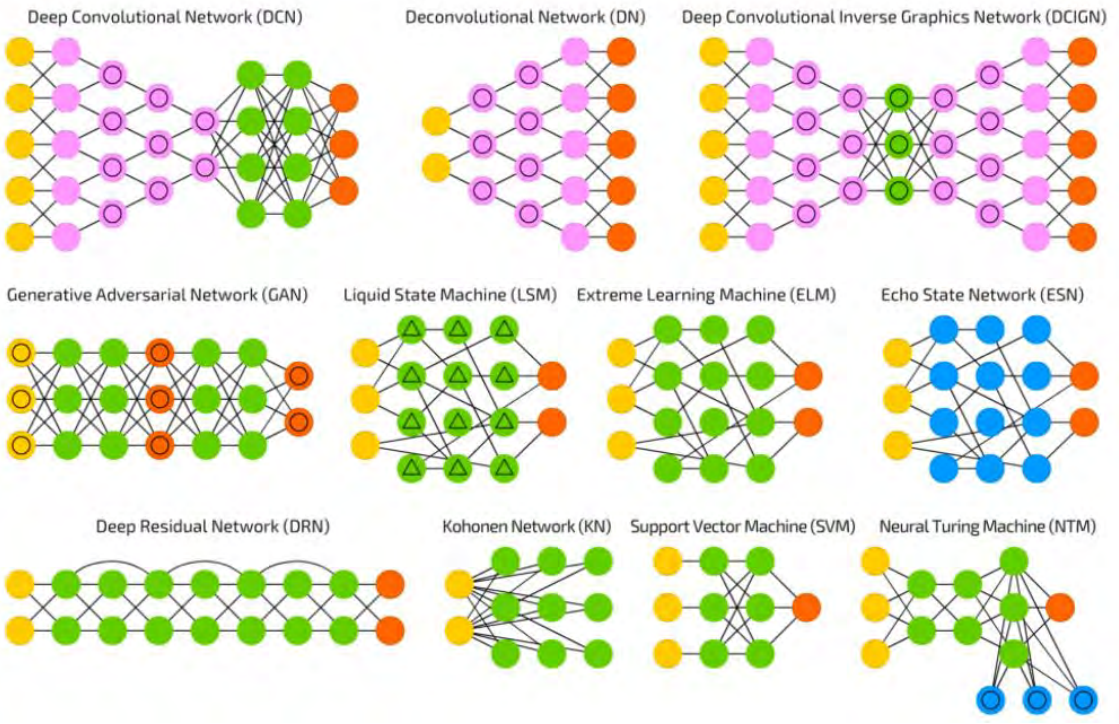


Рисунок 4.13 – Разновидность принципов работы нейронных сетей

Residual Neural Network (ResNet) – это тип глубокой нейронной сети, который стал популярным благодаря своей способности эффективно обучаться на глубоких архитектурах, содержащих сотни и даже тысячи слоев. Одной из ключевых идей ResNet является использование «соединений-остатков» (residual connections), которые позволяют обучать глубокие сети, минимизируя проблему затухания градиентов.

Принцип работы ResNet основан на использовании блоков с остаточными соединениями, известных как «Residual Blocks» (рис. 4.14).

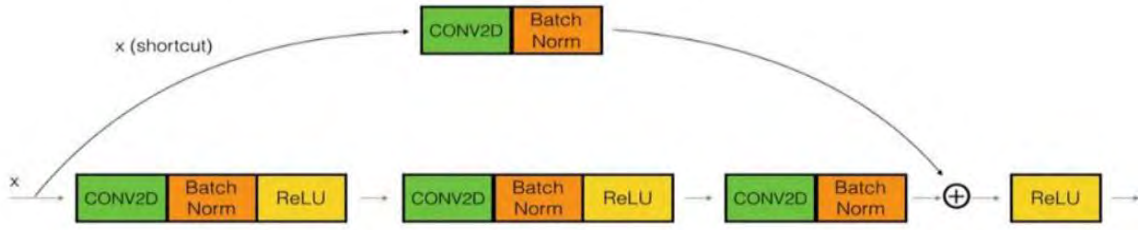


Рисунок 4.14 – Принцип работы ResNet

Обычные нейронные сети пытаются изучить отображения напрямую, например, входные данные могут быть преобразованы в более абстрактные представления с помощью последовательного стека слоев. Однако при этом могут возникать проблемы, такие как затухание градиентов, особенно в глубоких сетях.

ResNet предлагает решение этой проблемы, предполагая, что каждый блок сети должен изучать не исходное отображение, а разницу между исходным входом и его предсказанным значением. Это делается путем добавления остаточного (или «skip») соединения напрямую от входа блока к выходу. Таким образом, каждый блок изучает отклонение от исходного входа, что позволяет сети легче учиться представлять сложные функции и избегать затухания градиентов.

Преимущества ResNet включают в себя:

– обучение глубоких сетей. Благодаря остаточным соединениям ResNet позволяет эффективно обучать сети с глубокими архитектурами, содержащими десятки или сотни слоев;

– стабильность обучения. Остаточные соединения помогают предотвратить затухание градиентов и улучшить стабильность обучения;

– легкая восстановимость идентичности. Благодаря skip-connections сеть легче может научиться идентичным отображениям, что может быть полезно для решения некоторых задач, таких как восстановление изображений.

Использование ResNet стало широко распространено в области компьютерного зрения и обработки изображений, где глубокие сверточные сети играют важную роль в решении таких задач, как классификация изображений, обнаружение объектов и сегментация.

SegNet (семантической сегментации) – это архитектура нейронной сети, специально разработанная для решения задач семантической сегментации изображений, где каждый пиксель изображения классифицируется на определенный класс объекта. Принцип работы SegNet основан на энкодер-декодерной архитектуре с использованием сверточных нейронных сетей для извлечения признаков и последующего восстановления карты сегментации (рис. 4.15).

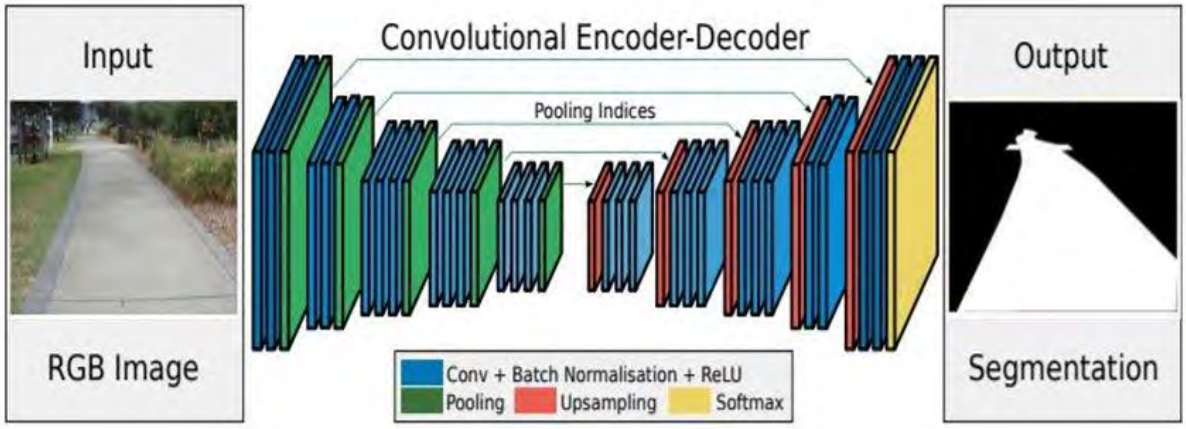


Рисунок 4.15 – Принцип работы SegNet

Процесс работы SegNet включает следующие этапы:

1. энкодирование (Encoder). В начале процесса изображение проходит через несколько слоев сверточных и пулинговых операций, что позволяет извлечь более абстрактные признаки из входного изображения. Это помогает сети понимать контекст и структуру изображения;

2. декодирование (Decoder). После прохождения через слои энкодера, представление изображения снижается по размеру. Декодерная часть архитектуры SegNet предназначена для восстановления пространственных размеров изображения и восстановления деталей с помощью операций деконволюции и интерполяции;

3. получение карты сегментации. На этом этапе сеть выдает карту сегментации, где каждый пиксель изображения отнесен к определенному классу объекта. Это достигается с помощью активационной функции на выходном слое, которая обычно является softmax для задачи сегментации, где каждый класс имеет свой собственный канал на выходе.

Преимущества SegNet:

– эффективное использование вычислительных ресурсов: благодаря энкодер-декодерной архитектуре сеть способна работать с высоким качеством сегментации, при этом требуя меньше вычислительных ресурсов, чем некоторые другие подходы;

– способность обрабатывать изображения разного размера: благодаря операциям пулинга и интерполяции SegNet может работать с изображениями разного размера без необходимости их предварительного изменения;

– применимость к различным задачам: SegNet может успешно применяться для решения различных задач сегментации, таких как сегментация объектов на дорожных сценах, сегментация медицинских изображений и другие.

U-Net остается одним из наиболее распространенных выборов для задач сегментации изображений благодаря своей эффективности и отличной производительности в различных областях, включая медицинское изображение, анализ снимков со спутников и обработку изображений в области компьютерного зрения (рис. 4.16).

SegNet и U-Net – это две архитектуры нейронных сетей, широко используемые в задачах сегментации изображений, таких как сегментация семантических объектов и медицинское сегментирование.

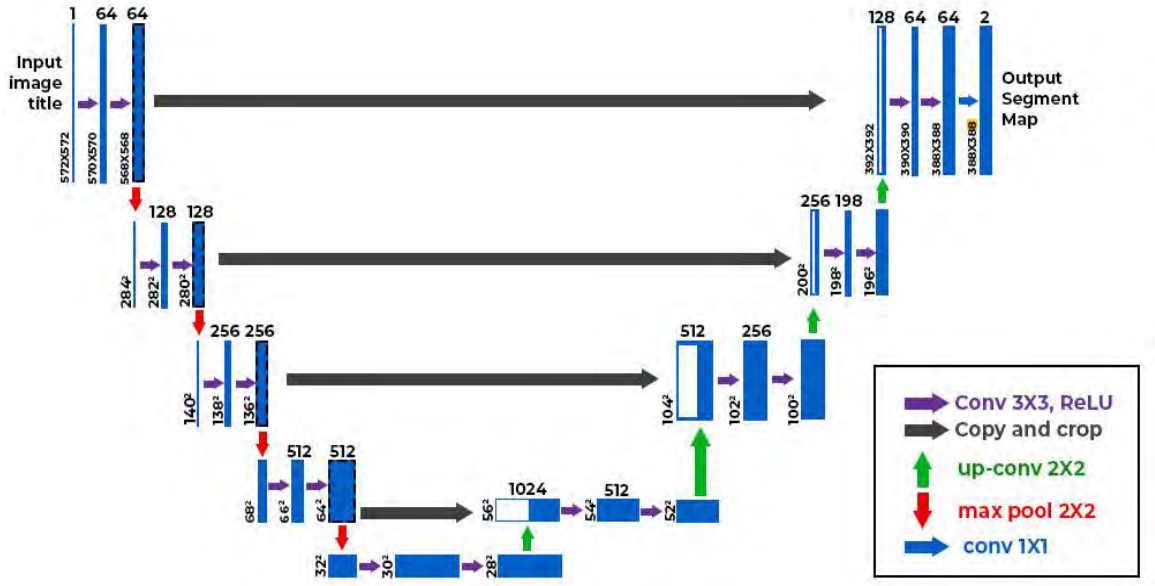


Рисунок 4.16 – Принцип работы U-Net

Обе архитектуры предлагают решения для извлечения информации о структуре объектов на изображении и применяются в различных областях, таких как медицинская диагностика, автоматическое вождение и анализ изображений.

Различия нейронных сетей SegNet и U-Net:

– архитектура. SegNet имеет более простую архитектуру с максимальным пулингом, в то время как U-Net имеет более сложную "U"-образную архитектуру с использованием skip-connections;

– принцип декодирования: В SegNet используется декодирование с помощью unpooling и деконволюции, тогда как U-Net использует декодирование с помощью skip-connections и апсемплинга;

– сохранение информации: U-Net сохраняет более детальную информацию о признаках за счет skip-connections, что позволяет более точно восстанавливать детали в сегментированном изображении.

Генеративно-состязательные сети (GAN) – это тип нейронных сетей, который представляет собой модель машинного обучения, включающую в себя две нейронные сети: генератор и дискриминатор. Они работают вместе в процессе обучения, соревнуясь друг с другом, чтобы достичь определенной цели, обычно связанной с созданием новых данных, которые могли бы быть идентичными по распределению некоторому набору реальных данных.

Принцип работы генеративно-состязательных сетей (GAN) основан на алгоритме машинного обучения без учителя, построенный на комбинации из двух нейронных сетей, одна из которых (сеть G) генерирует образы, а другая (сеть D) старается отличить правильные образы от неправильных.

$$\min_G \max_d J(G,d) = E\left[\ln(d(X))\right] + E\left[(\ln(1-d(G(z)))\right]. \qquad (4.6)$$

где, min_G – сеть G (генератор минимизирует и добивается того чтобы генератор ошибался),

max_d – сеть D (дискриминатор пытается максимизировать награду, пытается угадывать).

Генеративно-состязательная сеть состоит из двух нейронных сетей – генератора и дискриминатора, которые соревнуются между собой, чтобы достичь определенных целей (рис. 4.17).

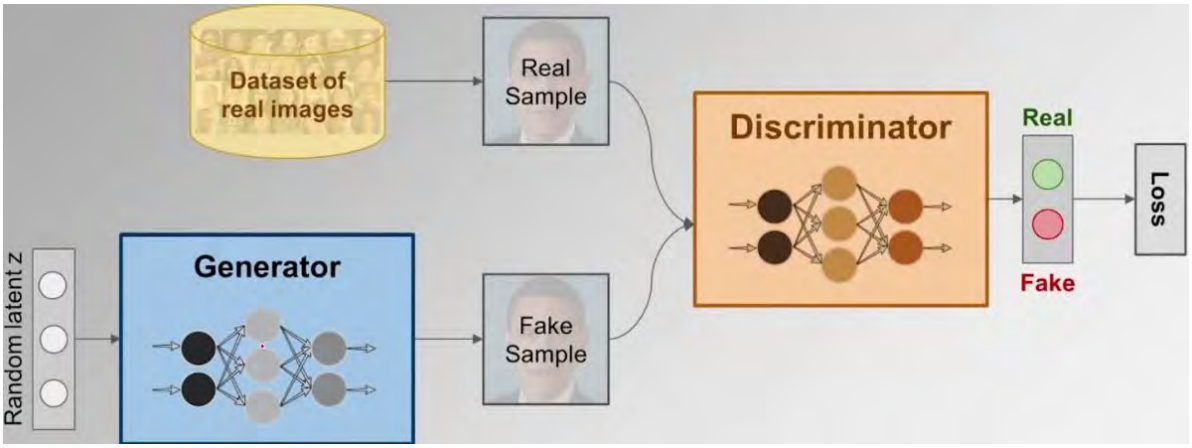


Рисунок 4.17 - Принцип работы генеративно-состязательной сети (GAN)

Генератор принимает на вход случайный шум или другой входной сигнал и стремится создать данные, которые выглядят как реальные. Его цель – обучиться генерировать данные, которые максимально приближены к реальным, минимизируя ошибку между сгенерированными данными и реальными данными.

Дискриминатор, с другой стороны, принимает на вход данные и пытается определить, являются ли они реальными (принадлежат ли реальному

набору данных) или сгенерированными генератором. Его цель – обучиться классифицировать данные, минимизируя ошибку в определении подлинности данных.

В процессе обучения генератор и дискриминатор играют в минимаксную игру, где генератор стремится создавать все более реалистичные данные, а дискриминатор стремится становиться все более точным в различении реальных данных от сгенерированных. Это приводит к динамике, в результате которой генератор становится все лучше в обмане дискриминатора, а дискриминатор становится все более точным в различении сгенерированных данных от реальных.

Постепенно обучение приводит к сходимости генератора и дискриминатора к равновесному состоянию, где генерируемые данные становятся практически неотличимыми от реальных данных. Этот процесс требует тщательной настройки гиперпараметров и архитектуры сетей, чтобы достичь стабильного обучения и качественных результатов.

Применение генеративно-состязательная нейроной сети:

– генерация изображений. GAN могут использоваться для генерации фотореалистичных изображений, лиц, архитектурных чертежей и многого другого;

– улучшение изображений. Они могут использоваться для улучшения качества изображений, увеличения разрешения, удаления шума и других артефактов;

– генерация текста и звука. GAN могут использоваться для генерации текста, аудиозаписей и других типов данных.

Преимущества генеративно-состязательная нейроной сети:

– создание реалистичных данных. GAN могут генерировать данные, которые выглядят и звучат очень реалистично;

– многообразие. Нейроные сети данного типа способны генерировать много различных вариантов данных, что делает их полезными в различных задачах;

– состязательный процесс обучения позволяет GAN создавать данные, которые сложно отличить от реальных, так как они обучаются улучшать свои навыки в процессе соперничества.

Несмотря на свои преимущества, GAN также могут сталкиваться с проблемами, такими как нестабильность обучения, режим коллапса и проблемы с оценкой качества созданных данных. Однако они остаются мощным инструментом для создания новых данных и улучшения существующих.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Адаменко, А. Н. Логическое программирование и Visual Prolog / А. Н. Адаменко, А. М. Кучуков. – Санкт-Петербург : БХВ – Петербург, 2003. – 992 с. – Текст : непосредственный.
2. Андрейчиков, А. В. Интеллектуальные информационные системы: учебник / А. В. Андрейчиков, О. Н. Андрейчикова. – Москва : Финансы и статистика, 2006. – 424 с. – Текст : непосредственный.
3. Барский, А. Б. Искусственный интеллект и интеллектуальные системы управления : монография / А. Б. Барский. – Москва : РУСАЙНС, 2024. – 186 с. – Текст : непосредственный.
4. Борисов, В. В. Нечёткие модели и сети / В. В. Борисов, В. В. Круглов, А. С. Федулов. – Москва : Горячая линия– Телеком, 2007. – 284 с. – Текст : непосредственный.
5. Братко, И. Алгоритмы искусственного интеллекта на языке Prolog / И. Братко. – Москва : Вильямс, 2004. – 637 с. – Текст : непосредственный.
6. Вакуленко, С. А. Практический курс по нейронным сетям / С. А. Вакуленко, А. А. Жихарева. – Санкт-Петербург : Университет ИТМО, 2018. – 71 с. – Текст : непосредственный.
7. Гифт, Н. Прагматичный ИИ. Машинное обучение и облачные технологии : науч. изд. / Н. Гифт ; пер. с англ. И. Пальти. – Санкт-Петербург : Питер, 2019. – 300 с. – Текст : непосредственный.
8. Головко, В. А. Нейросетевые технологии обработки данных : учеб. пособие / В. А. Головко, В. В. Краснопрошин. – Минск : БГУ, 2017 – 263 с. – Текст : непосредственный.
9. ГОСТ Р 59276–2020. Системы искусственного интеллекта. Способы обеспечения доверия. Общие положения. национальный стандарт Российской Федерации : издание официальное : утв. и введ. в действие приказом Федерального агентства по техническому регулированию и метрологии от 23 декабря 2020 г. № 1371-ст : введ. впервые : дата введ. 2021-03-01 / разработан Акционерным обществом «Всероссийский научно-исследовательский институт сертификации», Обществом с ограниченной ответственностью «ТВпортал». – Москва : Стандартинформ, 2021. – 16 с. – Текст : непосредственный.
10. ГОСТ Р 59277–2020. Системы искусственного интеллекта. Классификация систем искусственного интеллекта : национальный стандарт Российской Федерации : издание официальное : утв. и введ. в действие приказом Федерального агентства по техническому регулированию и метрологии от 23 декабря 2020 г. № 1372-ст : введ. впервые : дата введ. 2021-03-01 / разработан Акционерным обществом «Всероссийский научно-исследовательский институт сертификации», Обществом с ограниченной ответственностью «ТВпортал». – Москва : Стандартинформ, 2021. – 16 с. – Текст : непосредственный.

11. Гудфеллоу, Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль ; пер. с анг. А. А. Слинкина. – Изд. 2-е, испр. – Москва : ДМК Пресс, 2018. – 652 с. – Текст : непосредственный.

12. Дейвенпорт, Т. Внедрение искусственного интеллекта в бизнес-практику: преимущества и сложности : учебник / Т. Дейвенпорт ; пер. с англ. З. Мамедьянова. – Москва : Сбербанк, 2019. – 250 с. – Текст : непосредственный.

13. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс. – Москва : ДМК-Пресс, 2006. – 312 с. – Текст : непосредственный.

14. Иванов, В. М. Интеллектуальные системы : учебное пособие / В. М. Иванов. – Екатеринбург : Изд-во Урал. ун-та, 2015. – 92 с. – Текст : непосредственный.

15. Интеллектуальные информационные системы и технологии : учебное пособие / Ю. Ю. Громов, О. Г. Иванова, В. В. Алексеев [и др.]. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2013. – 244 с. – Текст : непосредственный.

16. Исследование перспектив и проблем интеграции человека с компьютером: искусственный интеллект, робототехника, технологическая сингулярность и виртуальная реальность / С. А. Васюгова, А. В. Остроух, М. Н. Краснянский, А. Самаратунга. – Текст : непосредственный // Перспективы науки. – 2011. – № 4 (19). – С. 109-112.

17. Косаренко, Н. Н. Система искусственного интеллекта: понятие, теория, право и перспективы развития : монография / Н. Н. Косаренко. – Москва : РУСАЙНС, 2024. – 176 с. – Текст : непосредственный.

18. Кузнецов, А. В. Искусственный интеллект и информационная безопасность общества : монография / А. В. Кузнецов, С. И. Самыгин, М. В. Радионов. – Москва : РУСАЙНС, 2024. – 118 с. – Текст : непосредственный.

19. Мухамедиев, Р. И. Введение в машинное обучение: учебник / Р. И. Мухамедиев, Е. Н. Амиргалиев. – Алматы, 2022. – 252 с. – Текст : непосредственный.

20. О развитии искусственного интеллекта в Российской Федерации : указ Президента Российской Федерации № 490 : утверждён 10.10.2019. – Текст : электронный // Правительство России официальный сайт. – 2024. – URL : http://government.ru/docs/all/124098/ (дата обращения : 07.10.2024).

21. Осовский, С. Нейронные сети для обработки информации / С. Осовский ; пер. с пол. И. Д. Рудинского. – Москва: Финансы и статистика, 2002. – 344 с. – Текст : непосредственный.

22. Остроух, А. В. Введение в искусственный интеллект : монография / А. В. Остроух. – Красноярск : Научно-инновационный центр, 2020. – 250 с. – Текст : непосредственный.

23. Остроух, А. В. Интеллектуальные системы в науке и производстве / А. В. Остроух, А. Б. Николаев. – Saarbrucken, Germany: Palmarium Academic Publishing, 2012. – 312 с. – Текст : непосредственный.

24. Остроух, А. В. Основы построения систем искусственного интеллекта для промышленных и строительных предприятий: монография / А. В. Остроух. – Москва: ООО «Техполиграфцентр», 2008. – 280 с. – Текст : непосредственный.

25. Остроух, А. В. Системы искусственного интеллекта в промышленности, робототехнике и транспортном комплексе: монография / А. В. Остроух. – Красноярск : Научно-инновационный центр, 2013. – 326 с. – Текст : непосредственный.

26. Остроух, А. В. Системы искусственного интеллекта: монография / А. В. Остроух, Н. Е. Суркова. – Санкт-Петербург : Лань, 2019. – 228 с. – URL : https://e.lanbook.com/book/113401 (дата обращения: 07.10.2024). – Текст : электронный.

27. Ростовцев, В. С. Искусственные нейронные сети : учебник для вузов / В. С. Ростовцев. – 2-е изд., стер. – Санкт-Петербург : Лань, 2021. – 2016 с. – Текст : непосредственный.

28. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечёткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский ; пер. с польск. И. Д. Рудинского. – Москва: Горячая линия – Телеком, 2007. – 452 с. – Текст : непосредственный.

29. Рыбина, Г. В. Основы построения интеллектуальных систем : учеб. пособие / Г. В. Рыбина. – Москва : Финансы и статистика : ИНФРА-М, 2010. – 430 с. – Текст : непосредственный.

30. Сергеев, А. П. Введение в нейросетевое моделирование : учеб. пособие / А. П. Сергеев, Д. А. Тарасов. – Екатеринбург : Изд-во Урал. ун-та, 2017. – 128 с. – Текст : непосредственный.

31. Тьюринг, А. Может ли машина мыслить. Общая и логическая теория автоматов / А. Тьюринг, Дж. Нейман ; пер. с англ. Ю. А. Данилова. – Москва: URSS, 2019. – 232 с. – Текст : непосредственный.

32. Фисун, В. В. Искусственный интеллект управления информационной безопасностью объектов критической информационной структуры : монография / В. В. Фисун. – Москва : РУСАЙНС, 2023. – 360 с. – Текст : непосредственный.

33. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – 2-е изд., испр. – Москва : Санкт-Петербург : Диалектика, 2019. – 1103 с. – Текст : непосредственный.

34. Черняк, Е. Введение в глубокое обучение / Е. Черняк ; пер. с англ. – Санкт-Петербург : ООО «Диалектика», 2020. – 192 с. – Текст : непосредственный.

35. Эртель, В. Введение в искусственный интеллект / В. Эртель ; пер. с англ. А. Горман. – Москва : Эксмо, 2019. – 448 с. – Текст : непосредственный.

36. Ясницкий, Л. Н. Введение в искусственный интеллект : учебное пособие / Л. Н. Ясницкий. – Москва : Academia, 2005. – 176 с. – Текст : непосредственный.

Учебное издание

Гаваев Александр Сергеевич
Лысенко Игорь Вячеславович
Чайников Денис Анатольевич
Губенко Арсений Сергеевич

ОСНОВЫ СИСТЕМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

В авторской редакции

Подписано в печать 11.12.2024. Формат 60x90 1/16. Усл. печ. л. 5,5.
Тираж 500 экз. Заказ № 2955.

Библиотечно-издательский комплекс
федерального государственного бюджетного образовательного
учреждения высшего образования
«Тюменский индустриальный университет».
625000, Тюмень, ул. Володарского, 38.

Типография библиотечно-издательского комплекса.
625039, Тюмень, ул. Киевская, 52.