**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет инфокоммуникационных сетей и систем (ИКСС)

Курсовая работа по теме «Разработка нейронной сети для определения десятичных цифр рукописного ввода по изображению»

по курсу

«Операционные системы и сети»

Группа: ИКПИ-14

Выполнил студент: Хохлов Т. В.

Принял преподаватель: Дагаев А.В.

Место для подписи \_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2023 г.

Оглавление

[**Нейронная сеть (Neural Network)** 3](#_Toc156435265)

[**Связь с машинным зрением** 3](#_Toc156435266)

[**Что такое нейрон?** 4](#_Toc156435267)

[**Что такое синапс?** 6](#_Toc156435268)

[**Как работает нейронная сеть?** 7](#_Toc156435269)

[**Функция активации или слой активации** 8](#_Toc156435270)

[**Функция активации ReLU** 9](#_Toc156435271)

[**Функция активации Softmax** 10](#_Toc156435272)

[**Тренировочные сеты. Набор данных MNIST.** 12](#_Toc156435273)

[**Разработка нейронной сети.** 13](#_Toc156435274)

[**Пример работы программы.** 18](#_Toc156435275)

[**Алгоритм выполнения программы.** 20](#_Toc156435276)

[Исходный код программы 20](#_Toc156435277)

# **Нейронная сеть (Neural Network)**

Нейронная сеть — это вычислительная модель, вдохновленная структурой и функцией нервной системы живых организмов. Она применяется в области машинного обучения для анализа данных, распознавания образов, и решения различных задач, от классификации до генерации контента. Для полного понимания структуры и работы нейронных сетей, нужно подробно рассмотреть на её составляющие элементы и параметры.

## **Связь с машинным зрением**

Машинное зрение — область машинного обучения, посвященная разработке методов и алгоритмов, позволяющих компьютерам анализировать и интерпретировать визуальную информацию, аналогично человеческому зрению.

Нейронные сети в машинном зрении применяются для распознавания образов на изображениях. Входные данные (изображения) подаются на входной слой сети, где они проходят через сверточные и пулинг слои, извлекая признаки. Далее, полносвязные слои и функции активации позволяют сети классифицировать или регрессировать данные.

В контексте машинного зрения, нейронные сети обучаются извлекать визуальные признаки, что позволяет им распознавать объекты, лица, и выполнять другие задачи визуального анализа. Эта способность нейронных сетей делает их мощным инструментом в сфере обработки изображений и расширяет возможности автоматизации и анализа визуальных данных.

## **Что такое нейрон?**



Рис. 1 – Схема составного персептрона

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа (рис. 1): входной (синий), скрытый (красный) и выходной (зеленый). В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Соответственно, есть входной слой, который получает информацию, n скрытых слоев (обычно их не больше 3), которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть 2 основных параметра: входные данные **(input data)** и выходные данные **(output data)**. В случае входного нейрона: input=output (рис. 2). В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации и попадает в поле output.

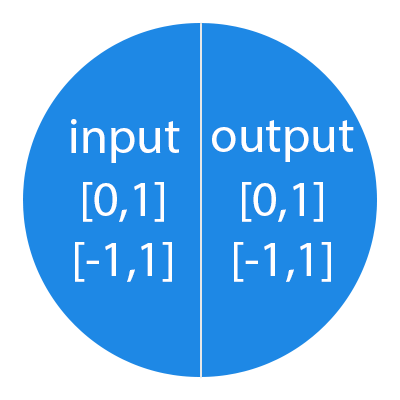


Рис. 2 – Структура входного нейрона

Нейроны оперируют числами в диапазоне [0,1] или [-1,1]. Для обработки чисел вне данного диапазона необходимо провести процесс нормализации (поделить 1 на необходимое число).

## **Что такое синапс?**

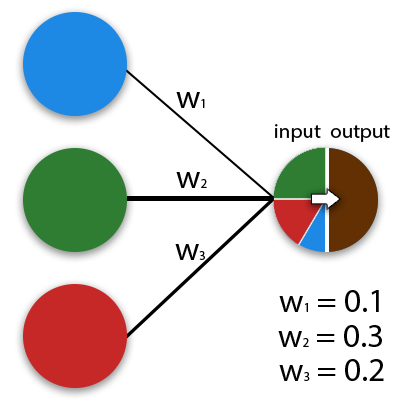


Рис. 3 – Схема примитивного персептрона

**Синапс** — это связь между двумя нейронами. У синапсов есть 1 параметр — **вес**. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Например, имеется 3 нейрона, которые передают информацию следующему. Тогда имеются 3 веса, соответствующие каждому из этих нейронов. Нейроны, связанные синапсом с наибольшим весом будут наиболее актуальными, то есть информация предыдущего нейрона будет содержать доминирую в следующем нейроне. На самом деле, совокупность весов нейронной сети или матрица весов — это своеобразный мозг всей системы. Именно благодаря этим весам, входная информация обрабатывается и превращается в результат.

Важно помнить, что во время инициализации нейронной сети, веса расставляются в случайном порядке.

## **Как работает нейронная сеть?**

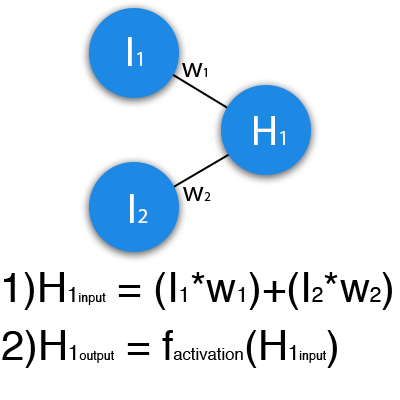


Рис. 3 – Схема примитивного персептрона, I-входной нейрон, H-скрытый нейрон.

Для простоты и полного понимая работы нейронных сетей, возьмём пример изображенный на рисунке 3. На нём изображён примитивный персептрон, состоящий из двух входных нейронов (I1 и I2) и одного скрытого (H1) соединённых весами w1 и w2. Из формулы 1 видно, что входная информация, переходящая на следующий слой — это сумма всех входных данных, умноженных на соответствующие им веса. В представленной формуле входные нейроны, представляют собой наличие определённой характеристики (или же степень привязанности) для рассматриваемого объекта, а веса – степень актуальности данной характеристики. Как было описано ранее, веса в только проинициализированной нейронной сети расставляются в случайном порядке, т.е. можно предположить, что у нас имеются какие-то входные данные (теоретические характеристики и проинициализированные веса), но как понять какие веса для определенной задачи будут актуальными? И ведь действительно, когда у нас есть входные данные, мы передаем их по слоям «вперёд», выполняя вычисления степени их актуальности и таким образом находим определённое значение, пока не дойдем до выходного нейрона. Запустив такую сеть в первый раз, мы увидим, что ответ далек от правильно, потому что сеть не натренирована. Из этого можно сделать логический вывод, что тренировка нейронных сетей, заключается в определении конкретных диапазонов значений весов для связующих входных и выходных нейронов. Этим занимается промежуточный этап – нормализация данных с помощью **функции активации** и их сохранение в модели нейронной сети.

## **Функция активации или слой активации**

Слои активации — это один из основных типов слоев, которые используются в нейронных сетях. Они представляют собой функцию, которая добавляет нелинейность к выходу предыдущего слоя. Это позволяет нейронной сети лучше моделировать сложные функции и более точно предсказывать результаты.

Слои активации принимают на вход результаты предыдущего слоя, называемые входом, и преобразуют их в выходное значение, которое передается следующему слою. Для этого они используют функцию активации, которая определяет, каким образом данные будут преобразованы.

В данной работе используются два вида функций активации, это **ReLU** и **SoftMax**.

## **Функция активации ReLU**

ReLU (Rectified Linear Unit) — это нелинейная функция активации, которая широко используется в глубоком обучении. Она преобразует входное значение в значение от 0 до положительной бесконечности. Если входное значение меньше или равно нулю, то ReLU выдает ноль, в противном случае - входное значение.

Математически ReLU определяется следующим образом:



где max - функция, возвращающая максимальное значение из двух.

Графически ReLU выглядит как линейная функция с нулевым отсечением на оси абсцисс в точке 0. Это значит, что функция имеет постоянный наклон во всех точках, кроме точки 0, где происходит отсечение.

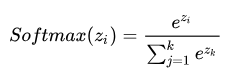


ReLU имеет несколько преимуществ по сравнению со сигмоидной функцией активации. Во-первых, ReLU более вычислительно эффективна, поскольку она является простой и быстрой операцией, которая не требует вычисления экспоненты. Во-вторых, ReLU решает проблему затухания градиента, так как она не вызывает затухания градиента при обратном распространении ошибки, как это происходит в случае с сигмоидной функцией активации.

## **Функция активации Softmax**

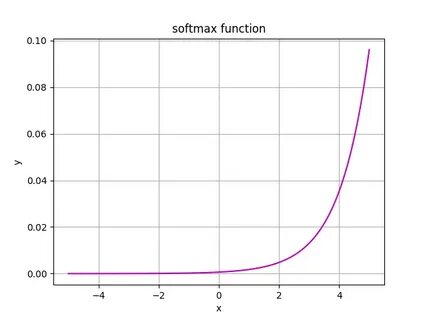
Функция Softmax используется для преобразования вектора значений в вероятностное распределение, которое суммируется до 1. Она особенно полезна в многоклассовой классификации, где необходимо определить вероятности для каждого класса.

Формула функции Softmax выглядит следующим образом:



где z\_i - это элемент входного вектора, а k - это общее число элементов в векторе.

График функции Softmax представляет собой гладкую кривую, начинающуюся от 0 и заканчивающуюся на 1, что соответствует сумме вероятностей всех элементов вектора. Кривая функции Softmax имеет свойство, что вероятность любого элемента вектора увеличивается, если значения других элементов уменьшаются, что позволяет использовать эту функцию для многоклассовой классификации.



Хотя функция Softmax имеет множество применений в машинном обучении, она также может иметь недостатки, такие как чувствительность к выбросам и несбалансированным данным, что может приводить к неверным вероятностным оценкам.

## **Тренировочные сеты. Набор данных MNIST.**

Как мы уже выяснили, чтобы решить задачу обучения нейронной сети, нужно посредством определенных математических алгоритмов определить верные значения весов синапсов связующих входные и выходные нейроны. Но какие входные данные предоставить?

Для нашей задачи необходимо предоставить созданной нейронной сети достаточное количество изображений для обучения. Так как определять нейронная сеть будет десятичные цифры рукописного ввода, то изображения должны быть представлены разного почерка, расположения цифры относительно центра изображения, а также размера цифры. Под данную характеристику отлично попадает **тренировочный сет MNIST**, который и будет использоваться в данной работе.

**База данных MNIST (сокращение от «Modified National Institute of Standards and Technology»)** — объёмная база данных образцов рукописного написания цифр. База данных является стандартом, предложенным Национальным институтом стандартов и технологий США с целью калибрации и сопоставления методов распознавания изображений с помощью машинного обучения в первую очередь на основе нейронных сетей. Данные состоят из заранее подготовленных примеров изображений, на основе которых проводится обучение и тестирование систем. База данных была создана после переработки оригинального набора чёрно-белых образцов размером 20x20 пикселей NIST. Создатели базы данных NIST, в свою очередь, использовали набор образцов из Бюро переписи населения США, к которому были добавлены ещё тестовые образцы, написанные студентами американских университетов. Образцы из набора NIST были нормализированы, прошли сглаживание и приведены к серому полутоновому изображению размером 28x28 пикселей.

База данных MNIST содержит 60000 изображений для обучения и 10000 изображений для тестирования. Половина образцов для обучения и тестирования были взяты из набора NIST для обучения, а другая половина — из набора NIST для тестирования.

# **Разработка нейронной сети.**

Для выполнения данной работы и создания нейронной сети был выбран язык программирования **«Python»**, поскольку он имеет большое количество инструментов для разработки нейронных сетей и как следствие - низкий порог входа для новичков.

За основу была взята библиотека машинного обучения с открытым исходным кодом – **TensorFlow,** разработанная компанией Google Brain *(исследовательское подразделение Google)* для создания и обучения разнообразных моделей машинного обучения.

Перед началом работы необходимо загрузить набор тренировочных данных. Следующая строка:

# Загрузка данных MNIST

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

загружает данные из набора MNIST и распаковывает их в переменные:

* **train\_images**: изображения, предназначенные для обучения модели.
* **train\_labels**: соответствующие метки классов (цифры от 0 до 9) для обучающих изображений.
* **test\_images**: изображения, предназначенные для тестирования модели.
* **test\_labels**: соответствующие метки классов для тестовых изображений.

Таким образом, после выполнения этой строки кода у нас будет доступ к обучающим и тестовым данным. Затем, поскольку изображения набора **MNIST** представляются в черно-белом формате размером 28x28, необходимо предварительно обработать полученные изображения:

# Предобработка данных

train\_images = train\_images.reshape((60000, 28, 28, 1)).astype('float32') / 255

test\_images = test\_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

В частности:

1. **train\_images.reshape((60000, 28, 28, 1))**: Обучающие изображения переформатируются так, чтобы каждое изображение имело размерность 28x28 пикселей и содержало всего один канал (в данном случае, черно-белый). Исходные изображения были в форме (60000, 28, 28), где 60000 - количество изображений, 28 - высота, и 28 - ширина.
2. **astype('float32') / 255**: Значения пикселей преобразуются в числа с плавающей запятой (float32) и нормализуются, деля на 255. Нормализация в диапазоне [0, 1] полезна для улучшения стабильности обучения нейронных сетей.

Аналогичные преобразования выполняются и для тестовых данных (**test\_images**). Таким образом, данные готовы для использования в обучении нейронной сети, которая ожидает входные данные в указанном формате и с нормализованными значениями пикселей.

Следующим шагом необходимо создать непосредственно модель нейронной сети:

# Создание модели

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

Этот код создает модель нейронной сети с использованием библиотеки **TensorFlow** и **Keras**. Давайте разберем каждый слой:

1. **Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1))**: Сверточный слой с 32 фильтрами размера (3, 3), функцией активации ReLU. Этот слой предназначен для извлечения признаков из входных изображений. **input\_shape=(28, 28, 1)** указывает, что входные изображения имеют размер 28x28 пикселей и содержат один канал (черно-белый).
2. **MaxPooling2D((2, 2))**: Слой пулинга, который уменьшает размерность данных, удаляя избыточные признаки. В данном случае используется максимальный пулинг с окном (2, 2).
3. **Conv2D(64, (3, 3), activation='relu')**: Еще один сверточный слой с 64 фильтрами.
4. **MaxPooling2D((2, 2))**: Еще один слой максимального пулинга.
5. **Conv2D(64, (3, 3), activation='relu')**: Третий сверточный слой.
6. **Flatten()**: Слой, преобразующий данные из трехмерного формата в одномерный. Необходим перед подачей данных в полносвязные слои.
7. **Dense(64, activation='relu')**: Полносвязный слой с 64 нейронами и функцией активации ReLU.
8. **Dense(10, activation='softmax')**: Выходной слой с 10 нейронами (по числу классов в задаче MNIST) и функцией активации softmax, которая используется для предсказания вероятностей принадлежности к каждому классу.

Эта архитектура представляет собой сверточную нейронную сеть (**CNN**) с несколькими сверточными слоями (**Conv2D**) для извлечения признаков из изображений и последующими полносвязными слоями (**Dense**) для классификации.

Следующий этап – скомпилировать созданную модель:

# Компиляция модели

model.compile(optimizer='adam',

            loss='categorical\_crossentropy',

            metrics=['accuracy'])

В процессе компиляции необходимо задать несколько параметров, которые определяют, как будет выполняться процесс обучения. Вот что означают параметры в данной строке:

1. **optimizer='adam'**: Оптимизатор, который будет использоваться для обновления весов модели и минимизации функции потерь. **"Adam" (Adaptive Moment Estimation)** — это эффективный метод стохастической оптимизации.
2. **loss='categorical\_crossentropy'**: Функция потерь, которую модель будет минимизировать в процессе обучения. В данном случае, поскольку решается задача классификации с несколькими классами, используется категориальная кросс-энтропия.
3. **metrics=['accuracy']**: Метрика, которую будет использовать модель для оценки своей производительности. Здесь используется точность (accuracy), которая измеряет долю правильных классификаций.

Таким образом, когда модель будет обучаться методом **fit**:

# Обучение модели

model.fit(train\_images, train\_labels, epochs=5, batch\_size=1, validation\_split=0.2)

с учётом параметров:

* **train\_images и train\_labels**: обучающие изображения и соответствующие им метки.
* **epochs=5**: Эпоха — это один проход через весь набор данных в обучении нейронной сети. В данном случае указано, что обучение будет проходить в течение 5 эпох.
* **batch\_size=1**: Размер пакета **(batch size)** определяет, сколько примеров данных используется для обновления весов модели в каждой итерации обучения. В данном случае, используется стохастический градиентный спуск с размером пакета 1, что означает, что обновление весов выполняется после каждого обучающего примера.
* **validation\_split=0.2**: Этот параметр указывает, какая часть обучающих данных должна быть использована для валидации (оценки производительности) в конце каждой эпохи. Здесь 20% данных будут использованы для валидации.

в процессе обучения модель будет прогонять обучающие данные через свои слои, вычислять потери, использовать оптимизатор для коррекции весов сети, и таким образом, постепенно настраивать модель для лучшей производительности на обучающих данных.

После обучения модели на тестовом сете мы наконец можем использовать её для определения десятичных чисел на необходимых нам изображениях.

# **Пример работы программы.**

**Задача:** необходимо с помощью нейросети определить рукописные десятичные числа на следующих изображениях.

**** ****    

**Пример выполнения поставленной задачи со следующими параметрами обучения:**

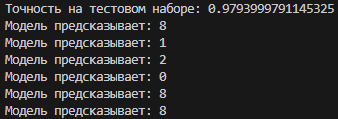
Количество эпох = 1;

Размер пакета = 1;

Часть валидации = 0.2;

**Результат работы:**



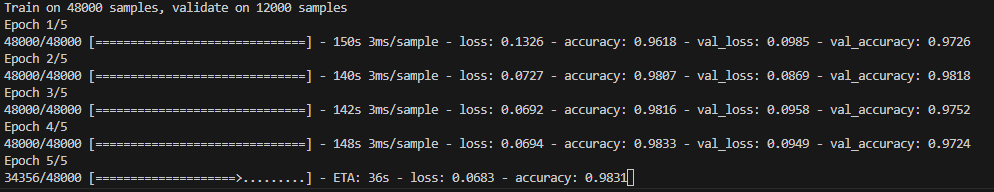


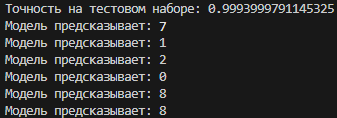
**Пример выполнения поставленной задачи со следующими параметрами обучения:**

Количество эпох = 5;

Размер пакета = 1;

Часть валидации = 0.2;





# **Алгоритм выполнения программы.**

1. **Создание и компиляция новой модели нейронной сети.**
2. **Обучение созданной модели на наборе данных MNIST.**
3. **Оценка модели на тестовом наборе данных.**
4. **Использование полученной модели для определения десятичных чисел на предоставленных изображениях.**

# Исходный код программы

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import cv2

import numpy as np

def imgParse(image\_path, model):

    # Загрузка и предобработка изображения

    img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        print(f"Ошибка загрузки изображения {image\_path}")

        return

    # threshold, перевод в черно-белое

    img = cv2.threshold(img, 128, 255, cv2.THRESH\_BINARY)[1]

    img = cv2.resize(img, (28, 28))

    img = img.reshape((1, 28, 28, 1)).astype('float32') / 255

    # Предсказание

    predictions = model.predict(img)

    predicted\_label = np.argmax(predictions[0])

    # Вывод предсказания

    print(f"Модель предсказывает: {predicted\_label}")

# Загрузка данных MNIST

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

# Предобработка данных

train\_images = train\_images.reshape((60000, 28, 28, 1)).astype('float32') / 255

test\_images = test\_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Кодирование меток в формат one-hot

train\_labels = tf.keras.utils.to\_categorical(train\_labels)

test\_labels = tf.keras.utils.to\_categorical(test\_labels)

# Создание модели

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

# Компиляция модели

model.compile(optimizer='adam',

            loss='categorical\_crossentropy',

            metrics=['accuracy'])

# Обучение модели

model.fit(train\_images, train\_labels, epochs=5, batch\_size=1, validation\_split=0.2)

# Оценка модели на тестовых данных

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f"Точность на тестовом наборе: {test\_acc}")

imgParse('test.jpg', model)

imgParse('test2.jpg', model)

imgParse('test3.jpg', model)

imgParse('test4.jpg', model)

imgParse('test5.jpg', model)

imgParse('test6.jpg', model)