МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

РАЗРАБОТКА ЭКСПЕРТНЫХ СИСТЕМ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Работу выполнил _____ П.Л. Бандуровский
(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
д-р. техн. наук, зав. каф., _____ А.В. Коваленко
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Г.В. Калайдина
(подпись)

Краснодар
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

КУРСОВАЯ РАБОТА

РАЗРАБОТКА ЭКСПЕРТНЫХ СИСТЕМ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Работу выполнил _____П.Л. Бандуровский
(подпись)

Направление подготовки 09.03.03 Прикладная информатика

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
д-р. техн. наук, зав. каф., _____ А.В. Коваленко
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Г.В. Калайдина
(подпись)

Краснодар
2022

РЕФЕРАТ

Курсовая работа 48с., 23 рис., 19 источников, 4 приложения.

СВЕРТОЧНАЯ НЕЙРОНАЯ СЕТЬ, ГЕНЕРАТИВНО-СОСТЯЗАТЕЛЬНАЯ НЕЙРОННАЯ СЕТЬ, КЛАССИФИКАЦИЯ, KERAS, PYTORCH, TANSERFLOW

Объектом исследования является разработка нейронных сетей с помощью Python и библиотек Keras и PyTorch.

Цель работы: разработка экспертных систем на основе нейронных сетей, а именно разработка нейронных сетей для таких задач как классификация и генерация данных.

В результате курсовой работы были изложены основные понятия, связанные с данной темой и рассмотрены методы для создания различных нейронных сетей. Были изучены основные принципы работы и созданы нейронные сети с различными архитектурами с использованием библиотек Keras и PyTorch.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С момента своего появления искусственные нейронные сети (ИНС) используются для решения различных прикладных задач. ИНС успешно применяются в широчайшем спектре приложений, таких как распознавание образов, прогнозирование, выявление зависимостей, сжатие данных, задачи управления и многие другие.

У нейронных сетей большой потенциал благодаря своей нелинейности. Это свойство делает нейронные сети более выгодными в использовании, нежели обычные программы. Во много это связано с тем, что у ИНС нет такой проблемы как проблема размерности.

В современном мире распознавание образов находит все большее применение в повседневной жизни людей. Методы и алгоритмы теории распознавания широко применяются в медицине (диагностика медицинских снимков), геологии (изучение природных ресурсов Земли), робототехнике (зрение роботов), астрономии, при анализах изображений, идентификации человека, автоматическом проектировании и т. д.

В связи с вышеизложенным, тема курсовой работы, направленная на изучение и разработку моделей нейронных сети, распознающей и генерирующей образы, является актуальной.

Актуальность данной работы обусловлена также широким применением подобных технологий и, вследствие этого, большой популярностью на рынке программ, распознающих и генерирующих образы.

Целью работы является разработка экспертных систем на основе нейронных сетей, а именно разработка регрессионной, классифицирующей и генерирующей нейронных сетей с использованием языка программирования Python, библиотек Keras и PyTorch. Данная цель определила следующие задачи:

– изучение теоретических основ нейронных сетей,

– изучение структуры нейронных сетей, правил их функционирования,

– рассмотрение основных типов задач, которые решают нейронные сети,

– теоретическое исследование алгоритмов обучения нейронных сетей,

– разработка нейронных сетей для классификации и генерации данных.

Структура курсовой работы состоит из трех глав, а также трех приложений, в которых представлен код нейронных сетей.

В первой главе курсовой работы представлен теоретический анализ предметной области, методы обучения нейронных сетей, рассмотрены основные типы задач нейросетевых технологий.

Во второй главе описаны основные программные средства реализации нейронных сетей, дана краткая характеристика языка программирования Python, описаны основные библиотеки, рабочая среда, обучающая выборка. Представлены различные архитектуры нейронных сетей, такие как генеративно-состязательные и сверточные нейронные сети. Описаны активационные функции, Dropout алгоритм оптимизации Адам.

В третьей главе представлена разработка классифицирующей, генерирующей и регрессионной нейронных сетей, созданных в курсовой работе: «Meteo», «Discernmentor» (первая и вторая версии), «Creator».

В заключение подведен основной итог работы.

1 Теоретический анализ нейронных сетей

Нейронные сети – вычислительные системы или машины, созданные для моделирования аналитических действий, совершаемых человеческим мозгом. Нейронные сети относятся к направлению искусственного интеллекта (ИИ) и применяются для распознавания скрытых закономерностей в необработанных данных, группировки и классификации, а также решения задач в области ИИ, машинного и глубокого обучения [1].

Работа нейронной сети сравнима с действиями человека: сталкиваясь с незнакомым предметом, он узнает его свойства и делает выводы. Аналогичные процессы происходят в узлах нейронных сетей, когда, решая определенную задачу, они используют полученный опыт для дальнейшего обучения.

Искусственные нейронные сети состоят из входных, скрытых и выходных слоев. Рисунок 1 схематично показывает расположение слоев, которые соединены связями (линиями)

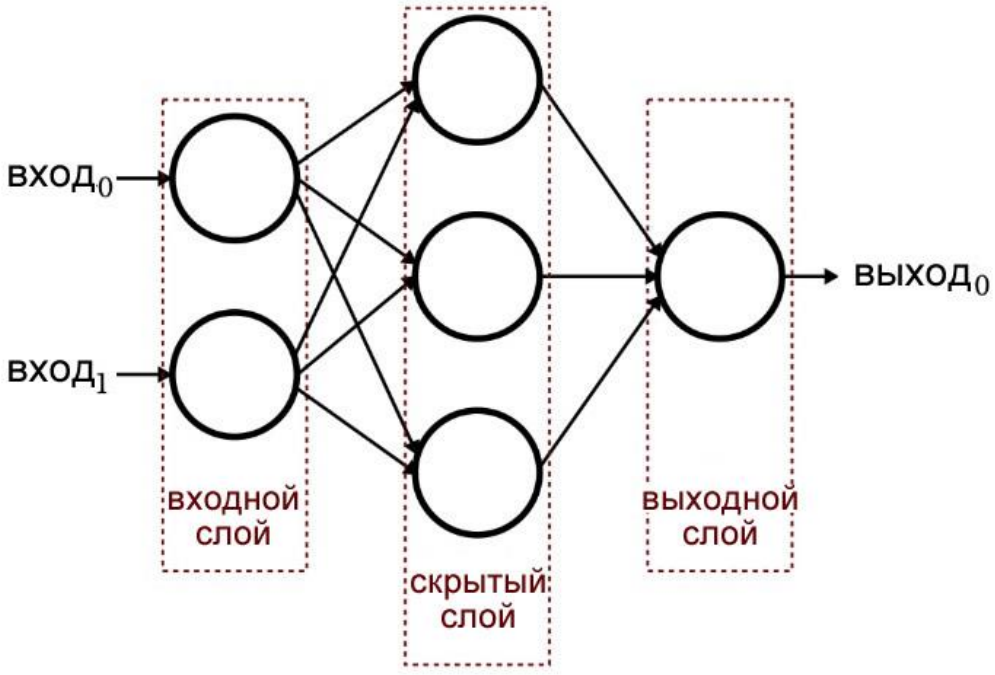

Рисунок 1 – Схема слоев нейронной сети [2]

В каждом из них есть несколько узлов, которые соединены со всеми узлами в сети с помощью разных связей и имеют свой «вес», влияющий на силу передаваемого сигнала. Такая архитектура позволяет вести параллельную обработку данных и постоянно сравнивать их с результатами обработки на каждом из этапов.

Основная единица нейронных сетей – искусственные нейроны, моделирующие основные функции биологических нейронов. Рисунок 2 показывает фундаментальное представление искусственного нейрона – перцептрона [3].

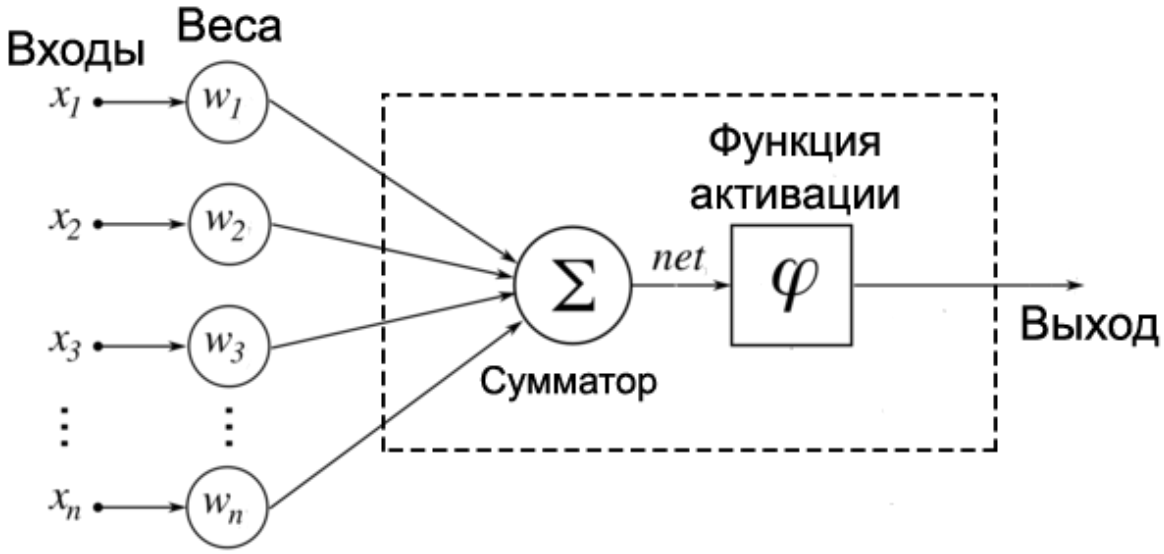

Рисунок 2 – Искусственный нейрон [4]

1.1 Методы обучения нейронных сетей

Нейронные сети изначально обучаются на размеченных наборах данных с очевидными закономерностями, а после используют полученные навыки для самообучения и достижения результата. При этом нейронная сеть может совершать миллионы попыток для достижения таких же результатов, как и предоставленном для обучения примере.

Существует два алгоритма обучения нейронных сетей:

1) Обучение нейронной сети с учителем – это метод преобразования одного набора данных в другой. Например, если представить, что имеется один набор данных «Котировки на бирже в понедельник», в котором записаны все котировки, имевшие место в каждый понедельник в течение последних 10 лет, и второй набор «Котировки на бирже во вторник» с котировками за тот же период, то алгоритм машинного обучения с учителем может попытаться использовать первый, чтобы предсказать второй. На рисунке 3 представлена нейронная сеть с учителем.

Рисунок 3 – Схематичное представление обучение нейронной сети с учителем [4]

2) Обучение нейронной сети без учителя – один из способов, при котором испытуемая система спонтанно обучается выполнять поставленную задачу без вмешательства со стороны экспериментатора. Как правило, это пригодно только для задач, в которых известны описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами. На рисунке 4 представлена нейронная сеть без учителя [4].

Рисунок 4 – Схематичное представление обучение нейронной сети без учителя [4]

Помимо этого, обучение нейронных сетей разделяются еще на два класса:

1) Параметрическое обучение. Параметрическая модель нейросети характеризуется наличием фиксированного числа параметров.

2) Непараметрическое обучение. Данная модель имеет бесконечное число параметров и определяется данными.

В процессе обучения нейронных сетей используется выборка, валидационная и тестовая выборки.

Обучающая выборка – это набор данных, который используется для разработки модели машинного обучения [5].

Валидационная выборка – это набор данных, который используется в процессе разработки модели машинного обучения для подбора оптимального набора гиперпараметров.

Тестовая выборка – это набор данных, который не используется непосредственно в процессе обучения модели или для подбора гиперпараметров, однако позволяет протестировать модель и является контрольным.

Параметрические модели обучения с учителем – это модели, имеющие фиксированное число регуляторов (это параметрическая часть таких моделей), обучение которых происходит путем поворота регуляторов. Входные данные обрабатываются согласно углу поворота регуляторов и преобразуются в пред сказание.

В параметрическом обучении без учителя используется схожий подход. Давайте рассмотрим в общих чертах этапы такого обучения. По сути, обучение без учителя осуществляет группировку данных. В параметрическом обучении без учителя регуляторы используются для группировки данных. В этом случае обычно имеется несколько регуляторов по числу групп, каждый из которых отражает близость входных данных к конкретной группе (с некоторыми исключениями и нюансами не забывайте, что это всего лишь обобщенное описание).

Непараметрическое обучение – это класс алгоритмов, в которых число параметров зависит от данных (то есть не предопределено). Это позволяет

использовать методы, выполняющие некоторые вычисления и увеличивающие число параметров, исходя из числа признаков, выявленных в данных.

1.2 Типы задач, которые решают нейросети

Выделяют несколько базовых типов задач, для решения которых могут использоваться нейронных сетей:

а) Классификация – для распознавания лиц, эмоций, типов объектов: например, квадратов, кругов, треугольников. Также для распознавания образов, то есть выбора конкретного объекта из предложенного множества: например, выбор квадрата среди треугольников.

б) Регрессия – для определения возраста по фотографии, составления прогноза биржевых курсов, оценки стоимости имущества и других задач, требующих получения в результате обработки конкретного числа.

в) Кластеризация – для изучения и сортировки большого объема неразмеченных данных в условиях, когда неизвестно количество классов на выходе, то есть для объединения данных по признакам. Например, кластеризация применяется для выявления классов картинок и сегментации клиентов.

г) Генерация – для автоматизированного создания контента или его трансформации. Генерация с помощью нейронных сетей применяется для создания уникальных текстов, аудиофайлов, видео, раскрашивания черно-белых фильмов и даже изменения окружающей среды на фото.

Таким образом, нейронные сети актуальны на сегодняшний день. Искусственный интеллект внедряется во все сферы деятельности, поэтому тема курсовой работы, посвященная разработке экспертных систем на основе нейронных сетей, а именно разработке нейронных сетей для таких задач как классификация и генерация данных является актуальной.

2 Основные программные средства реализации нейронных сетей

2.1 Язык программирования Python

Python – один из самых популярных языков программирования, с помощью которого можно решать самые разные задачи. Именно поэтому он так распространен и для создания нейронных сетей

Язык позволяет разрабатывать сложные алгоритмы за короткое время. Его отличают простота, лаконичность и выразительность. Помимо этого, он позволяет производить быстрые вычисления.

Нейронные сети – преимущественно небольшие программы, но при этом существует необходимость часто изменять их, подбирая наилучшую архитектуру, предобработку данных и другие параметры. Именно на Python есть библиотеки для более простого написания нейронных сетей [6].

2.1.1 Основные библиотеки

Keras – это библиотека для языка программирования Python, которая предназначена для глубокого машинного обучения. Она позволяет быстрее создавать и настраивать модели – схемы, по которым распространяется и подсчитывается информация при обучении. Но сложных математических вычислений Keras не выполняет и используется как надстройка над другими библиотеками [7].

Keras с версии 2.3 – это надстройка над библиотекой TensorFlow, которая нужна для машинного обучения. TensorFlow выполняет все низкоуровневые вычисления и преобразования и служит своеобразным движком, математическим ядром. Keras же управляет моделями, по которым проходят вычисления.

До версии 2.3 Keras мог использовать в качестве движка вычислительные различные библиотеки. Но в новых версиях поддержка прекратилась, теперь библиотека работает только с TensorFlow.

Keras создавалась как гибкая модульная библиотека, которую легко настраивать и модифицировать. Она бесплатная, у нее открытый исходный код, который может посмотреть любой человек.

Keras имеет узкую специализацию. Это инструмент для специалистов по машинному обучению, которые работают с языком Python: именно его чаще всего используют благодаря удобству математических вычислений. Keras применяют разработчики, которые создают, настраивают и тестируют системы машинного обучения и искусственного интеллекта, в первую очередь нейронные сети.

PyTorch – это научный вычислительный пакет на основе Python, который использует мощности графических процессоров. Это также одна из предпочтительных исследовательских платформ глубокого обучения, созданная для обеспечения максимальной гибкости и скорости. Он известен тем, что обеспечивает две из наиболее высокоуровневых функций; а именно, тензорные вычисления с сильной поддержкой ускорения графического процессора и построение глубоких нейронных сетей [8].

Существует много библиотек Python, которые могут изменить Ваше представление о том, как глубокое обучение и искусственный интеллект выполняются, и PyTorch – это одна из таких библиотек. Одной из ключевых причин успеха PyTorch является то, что можно легко создавать модели нейронных сетей. Он по-прежнему остается молодым игроком по сравнению с другими его конкурентами, при этом он быстро набирает обороты [9].

2.2 Рабочая среда

В качестве рабочей среды была выбрана среда Google Colaboratory. Это бесплатная среда, чтобы писать код в jupyter notebook. Она функционирует по принципу облака, поэтому над одним проектом могут работать одновременно несколько человек. Программа предоставляет доступ к графическим процессорам GPU и TPU. Благодаря их мощности можно исследовать искусственный интеллект и развивать приложения на основе нейронных сетей [11].

Colab позволяет использовать в одном файле исполняемый код, html-разметку, картинки. Этими файлами можно делиться: разрешать просматривать, редактировать и оставлять комментарии для совместной работы. Сервис помогает и с другими задачами:

1) сортировать данные,
2) строить визуализации,
3) проводить машинное обучение,
4) создавать системы для big data,
5) составлять прогнозы,
6) писать руководства.

2.3 Обучающая выборка

В качестве обучающих данный для обучения нейронных сетей была использована база данных MNIST. MNIST dataset – это база данных, в которой хранятся образцы написания рукописных цифр. Она состоит из 70 тысяч картинок одинакового размера, где изображены написанные от руки цифры.

Название расшифровывается как Modified National Institute of Standards and Technology – «Модифицированная база данных Национального института

стандартов и технологий США». Официальный сайт базы данных находится на этом [12]

Институт занимается стандартизацией: он собрал большой набор образцов почерка, привел все изображения с цифрами к единообразному виду и отрегулировал их.

Образцы написания цифр взяты из результатов переписи населения. Позже к ним добавлялись другие варианты, например взятые из тестирований студентов.

Набор данных MNIST активно применяют в машинном обучении, в частности в создании и обучении нейронных сетей. Это такие цифровые модели, которые по структуре повторяют соединения нейронов в человеческом мозгу. Их можно обучать, но для этого нужны большие массивы данных – так программа «запомнит», как выглядит то, с чем она имеет дело. На рисунке 5 показаны примеры цифр.

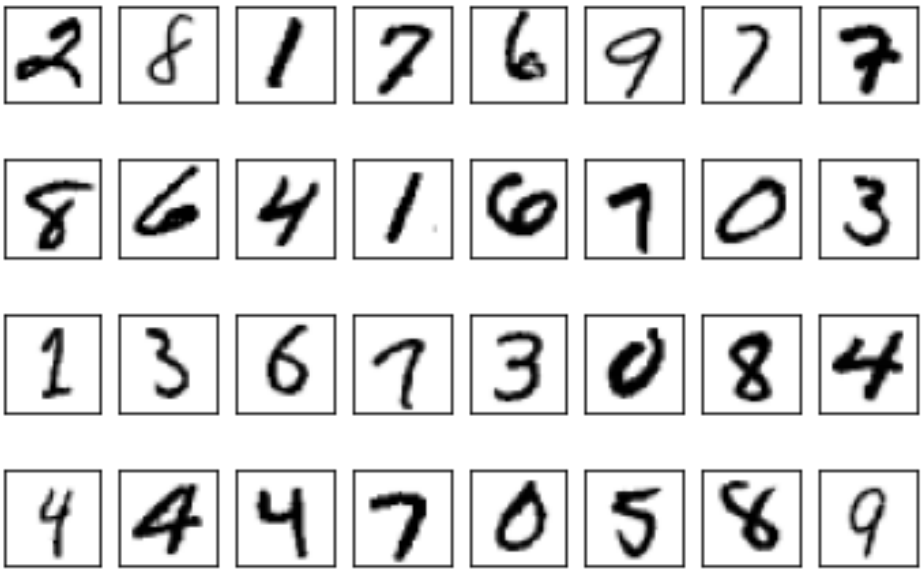

Рисунок 5 – Примеры цифр базы данных MNIST

Все картинки переведены в формат CSV и имеют размер 28×28 пикселей. У них черный фон, на котором изображена белая цифра. Цифра помещена в середине, так, чтобы ее центр масс совпадал с центром

изображения. Сама она чуть меньше целой картинки – ее размер составляет 20×20 пикселей. 70 тысяч картинок разделены на две части: train и test.

Train – это 60 тысяч изображений, которые должны использоваться для обучения. Программе показывают эти картинки, чтобы она «поняла», как выглядят те или иные цифры. В обучающем наборе уже есть правильные результаты, то есть программа сразу получает ответ, что именно ей показывают.

Test – это 10 тысяч изображений тестовой выборки. Их не используют для обучения, а показывают нейросети позже, чтобы проверить, насколько правильно она научилась распознавать числа [13].

2.4 Архитектуры нейронных сетей

Архитектура нейронной сети определяет общие принципы её построения. Конфигурация конкретизирует структуру сети в рамках заданной архитектуры: число нейронов, число входов и выходов сети, используемые активационные функции.

Различают следующие базовые архитектуры:

1) сети прямого распространения – все связи направлены строго от входных нейронов к выходным. К таким сетям относятся, например многослойный персептрон,

2) рекуррентные нейронные сети – сигнал с выходных нейронов или нейронов скрытого слоя частично передается обратно на нейроны входного слоя сети,

3) сети радиально-базисных функций – сети, содержащие единственны скрытый слой нейроны которого используют радиально-симметричную активационную функцию, применяются для решения задач классификации и прогнозирования,

4) полносвязные сети – нейронные сети, в которых каждый нейрон связан со всеми другими нейронами. Такие сети имеют самую высокую плотность связей.

2.5 Генеративно-состязательные нейросети

Генеративно-состязательная нейросеть (GAN) – архитектура, состоящая из генератора и дискриминатора, настроенных на работу друг против друга.

Потенциал GAN огромен, поскольку они имитируют любое распределение данных. GAN обучают создавать структуры, устрашающе похожие на сущности из нашего мира в области изображений, музыки, речи, прозы.

Дискриминационные алгоритмы пытаются классифицировать входные данные. Учитывая особенности полученных данных, они стараются определить категорию, к которой они относятся.

К примеру, пробегая все слова в письме дискриминационный алгоритм может предсказать, является сообщение спамом или не спамом. Спам – это категория, а пакет слов, собранный из электронной почты – образы, которые составляют входные данные. Математически категории обозначают y, а образы обозначают x. Запись p(y|x) используется для обозначения «вероятности y при заданном x», которая обозначает «вероятность того, что электронное письмо является спамом при имеющемся наборе слов».

Итак, дискриминационные функции сопоставляют образы с категорией. Они заняты только этой корреляцией.

Генеративные алгоритмы заняты обратным. Вместо того, чтобы предсказывать категорию по имеющимся образам, они пытаются подобрать образы к данной категории.

В то время как дискриминационные алгоритмы волнует взаимосвязь между y и x, генеративные алгоритмы позволяют находить p(x|y),

вероятность x при данном y или вероятность образов при данном классе (генеративные алгоритмы также могут использоваться в качестве классификаторов. Они могут делать больше, чем классифицировать входные данные.)

Процесс работы данной архитектуры заключается в следующем. Одна нейронная сеть, называемая генератором, генерирует новые экземпляры данных, а другая – дискриминатор, оценивает их на подлинность; т.е. дискриминатор решает, относится ли каждый экземпляр данных, который он рассматривает, к набору тренировочных данных или нет. На рисунке 6 представлена схема работы [14].

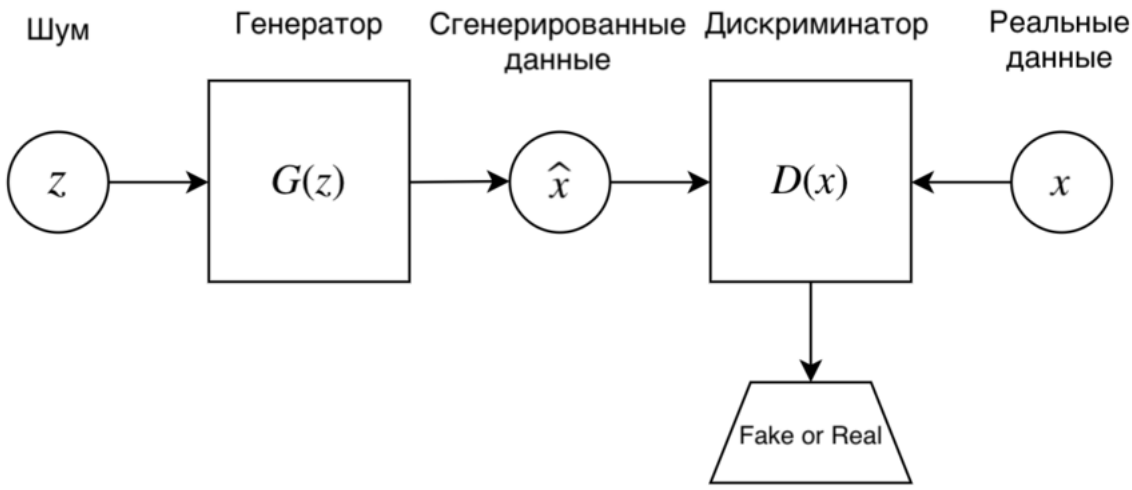

Рисунок 6 – Схема работы GAN архитектуры [15]

2.6 Сверточные нейронные сети

Это специальная архитектура нейронных сетей, нацеленная на эффективное распознавание изображений. В сверточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения (R-, G-, B-каналы

изображения). Основными видами слоев в сверточной нейронной сети являются сверточные слои (англ. convolutional layer), пулинговые слои (англ. pooling layer) и полносвязные слои (англ. fully-connected layer).

Сверточный слой нейронной сети представляет из себя применение операции свертки к выходам с предыдущего слоя, где веса ядра свертки являются обучаемыми параметрами. Еще один обучаемый вес используется в качестве константного сдвига (англ. bias).

В одном сверточном слое может быть несколько сверток. В этом случае для каждой свертки на выходе получится своё изображение.

Ядра свертки могут быть трёхмерными. Свертка трехмерного входа с трехмерным ядром происходит аналогично, просто скалярное произведение считается еще и по всем слоям изображения.

Можно заметить, что применение операции свертки уменьшает изображение. Также пиксели, которые находятся на границе изображения, участвуют в меньшем количестве сверток, чем внутренние. В связи с этим в сверточных слоях используется дополнение изображения (англ. padding). Выходы с предыдущего слоя дополняются пикселями так, чтобы после свертки сохранился размер изображения. Такие свертки называют одинаковыми (англ. same convolution), а свертки без дополнения изображения называются правильными (англ. valid convolution).

Еще одним параметром сверточного слоя является сдвиг (англ. stride). Хоть обычно свертка применяется подряд для каждого пикселя, иногда используется сдвиг, отличный от единицы – скалярное произведение считается не со всеми возможными положениями ядра, а только с положениями, кратными некоторому сдвигу.

Пулинговый слой призван снижать размерность изображения. Исходное изображение делится на блоки и для каждого блока вычисляется некоторая функция. Чаще всего используется функция максимума (англ. max pooling) или (взвешенного) среднего (англ. (weighted) average pooling). Обучаемых параметров у этого слоя нет. Основные цели пулингового слоя:

1) уменьшение изображения, чтобы последующие свертки оперировали над большей областью исходного изображения,

2) увеличение инвариантности выхода сети по отношению к малому переносу входа,

3)ускорение вычислений.

2.7 Активационные функции

В искусственных нейронных сетях функция активации нейрона определяет выходной сигнал, который определяется входным сигналом или набором входных сигналов. Рассмотрим несколько из активационных функций:

1) Сигмоида. Она нелинейна по своей природе, а комбинация таких функций производит тоже нелинейную функцию. Формула (1) является формулой сигмоиды

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

Еще одно достоинство такой функции – она не бинарна, что делает активацию аналоговой, в отличие от ступенчатой функции. Для сигмоиды также характерен гладкий градиент.

Если вы заметили, в диапазоне значений X от -2 до 2 значения Y меняется очень быстро. Это означает, что любое малое изменение значения X в этой области влечет существенное изменение значения Y [16].

2) Гиперболический тангенс. Это еще одна часто используемая активационная функция. Формула (2) является формулой гиперболического тангенса

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

Гиперболический тангенс очень похож на сигмоиду. И действительно, это скорректированная сигмоидная функция.

Поэтому такая функция имеет те же характеристики, что и у сигмоиды, рассмотренной ранее. Её природа нелинейна, она хорошо подходит для комбинации слоёв, а диапазон значений функции -(-1, 1). Поэтому нет смысла беспокоиться, что активационная функция перегрузится от больших значений.

3) Relu. Пользуясь формулой 3, становится понятно, что ReLu возвращает значение x, если x положительно, и *0* в противном случае

$$f(x) = \max(0, x) \quad (3)$$

ReLu нелинейна по своей природе, а комбинация ReLu также нелинейна! (На самом деле, такая функция является хорошим аппроксиматором, так как любая функция может быть аппроксимирована комбинацией ReLu).

ReLu менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции. Поэтому имеет смысл использовать ReLu при создании глубоких нейронных сетей [17].

2.8 Dropout

Dropout – это метод борьбы с переобучением нейронной сети. Цель этого метода – снизить специализацию каждого отдельного нейрона и сделать из них «специалистов более широкого профиля». Именно в этом корень проблемы переобучения. На каждой итерации изменения весовых коэффициентов часть нейронов нужно исключать с заданной вероятностью p. На рисунке 7 продемонстрирована схема работы метода dropout.

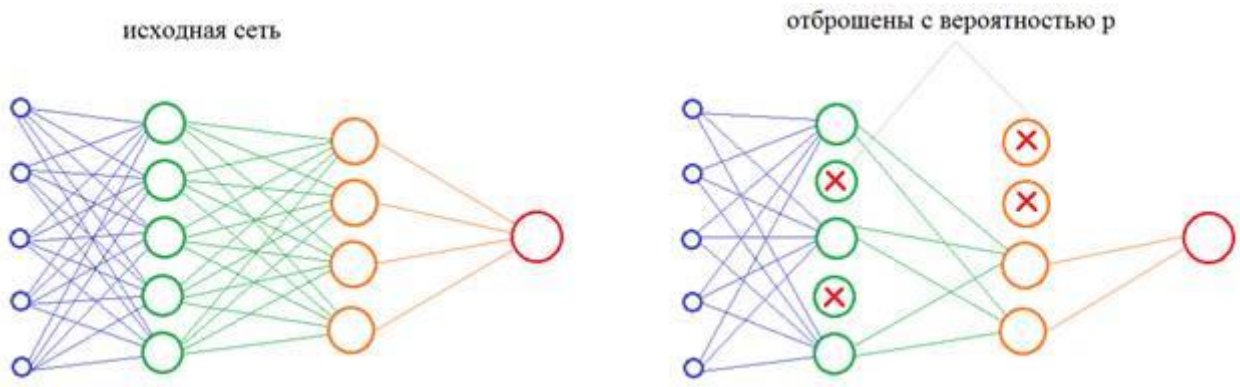

Рисунок 7 – Схема работы dropout [18]

В алгоритме dropout нейроны то выключаются, то включаются. После того, как сеть обучена, включаются все нейроны и эффект переобучения (излишней специализации) должен заметно снизиться [18].

2.9 Алгоритм оптимизации Адам

Адам – это алгоритм оптимизации, который можно использовать вместо классической процедуры стохастического градиентного спуска для итеративного обновления весов сети на основе обучающих данных.

Вместо того чтобы адаптировать скорости обучения параметров на основе среднего первого момента (среднего), Адам использует среднее значение вторых моментов градиентов (нецентрированная дисперсия).

В частности, алгоритм вычисляет экспоненциальную скользящую среднюю градиента и квадрата градиента, а параметры β_1 и β_2 управляют скоростью затухания этих скользящих средних.

Начальное значение скользящих средних и значений β_1 и β_2, близких к 1,0 (рекомендуется), приводит к смещению оценок моментов в сторону нуля. Это смещение преодолевается сначала вычислением смещенных оценок, а затем вычислением оценок с поправкой на смещение.

Адам является популярным алгоритмом в области глубокого обучения, потому что он быстро достигает хороших результатов.

Эмпирические результаты показывают, что Адам хорошо работает на практике и выгодно отличается от других методов стохастической оптимизации.

В оригинальной статье Адам был продемонстрирован эмпирически, чтобы показать, что конвергенция соответствует ожиданиям теоретического анализа Адам был применен к алгоритму логистической регрессии в наборах данных распознавания символов MNIST и IMDB, многослойном алгоритме персептрона в наборе данных MNIST и сверточных нейронных сетях в наборе данных распознавания изображений CIFAR-10.

Используя большие модели и наборы данных, Адам может эффективно решать практические проблемы глубокого обучения [19].

Таким образом, язык программирования Python удобен для разработки нейронных сетей, так как содержит мощные алгоритмы для нейросетей, удобные библиотеки.

3 Разработка классифицирующей, генерирующей и регрессионной нейронных сетей

В качестве результата курсовой работы были разработаны следующие нейронные сети:

1) Нейронная сеть «Meteo». В качестве входных данных принимает градусы в Цельсиях, а в качестве ответа переводит их в градусы Фаренгейта.

2) Нейронная сеть «Discernmentor». Программа способна распознавать рукописные цифры.

3) Нейронная сеть «Creator». Программа, которая способна сама генерировать рукописные цифры.

3.1 Нейронная сеть «Meteo»

Нейронная сеть была реализована с помощью пакета Keras и переводит градусы Цельсия в градусы Фаренгейта. Общая формула перевода представлена формулой 4.

$$F = 1{,}8 * C + 32 \quad (4)$$

где F – градусы Фаренгейта, C – градусы Цельсия.

В качестве обучающей выборки поступает массив input_data, который содержит градусы Цельсия и массив output_data, который содержит Градусы Фаренгейта. Нейронная сеть была обучена 600 эпох и имеет график ошибок, изображенный на рисунке 8, где по оси X отображено количество эпох, а по оси Y величина ошибки.

Структура нейронной сети состоит из двух входных нейронов и одного выходного нейрона, и она представлена на рисунке 8, где x соответствует градусам Цельсия, bias это смещение и равен 32 из формулы 6, ω1 и ω2 это

весовые коэффициенты, а y это градусы Фаренгейта, которые предсказывает нейронная сеть.

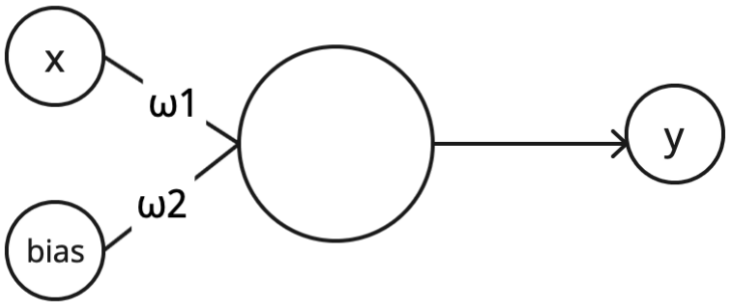

Рисунок 8 – Структура нейронной сети «Meteo»

Нейронная сеть состоит из 2 слоев – входного и выходного. Используется линейная активационная функция, оптимизатор Адам, а для поиска ошибки используется функция среднеквадратичной ошибки. Модель нейронной сети на Python представлена на рисунке 9.

```
model = keras.Sequential()
model.add(Dense(units = 1,
                input_shape=(1,),
                activation = 'linear'))
model.compile(loss = 'mean_squared_error',
              optimizer = keras.optimizers.Adam(0.1),
              metrics=['accuracy'])
```

Рисунок 9 – Модель нейронной сети «Meteo» на Python

После 1000 эпох обучения функция вычисления ошибки стала равна 0,064. Результаты вычисления ошибки представлены на рисунке 10, где ось x соответствует эпохе, обучения, а ось y соответствует вычисленной ошибке.

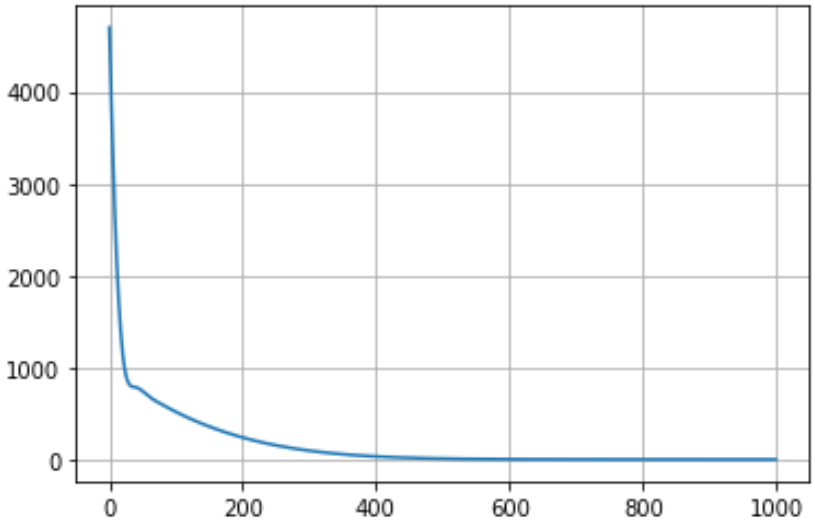

Рисунок 10 – График ошибок нейронной сети «Meteo»

В процессе работы программы мы вводим температуру в Цельсиях, а в качестве результата работы получаем температуру в Фаренгейтах. Для удобства сравнения результатов программа выводит и температуру, рассчитанную по формуле. На рисунке 11 показан пример работы нейронной сеть для 100 градусов.

```
Введите температуру в Цельсиях:
100
Температура в Фаренгейтах (предсказанная нейросетью) = 211.7422332763672
Температура в Фаренгейтах (рассчитанная по формуле) = 212.0
```

Рисунок 11 – Результат работы программы “Meteo”

Полный код программы представлен в приложении А.

3.2 Нейронная сеть «Discernmentor»

Данная нейронная сеть реализована в двух версиях. Рассмотрим первую версию.

3.2.1 Первая версия

Программа распознает рукописные цифры. Для обучения была взята выборка MNIST и была разделена на обучающую и валидационную выборки в соотношении 80/20.

После обучения нейронной сети точность определения рукописной цифры на обучающей выборке составляла 99.74%, а на тестовой 97.52%.

Это полносвязная нейронная сеть, модель которой состоит из входного, скрытого и выходного слоя, вход которой имеет 784 нейрона, скрытый слой имеет 128 нейрона, а выход состоит из 10 нейронов. Модель нейронной сети показана на рисунке 12.

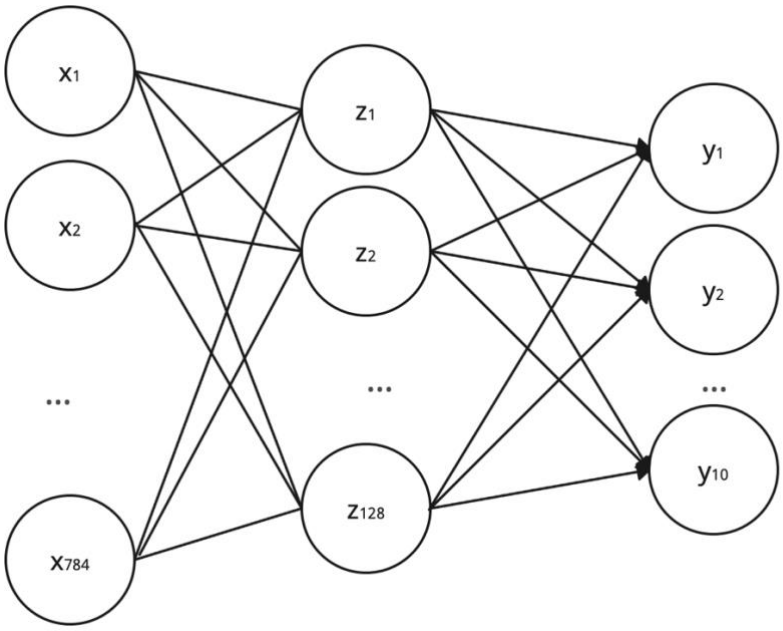

Рисунок 12 – Модель первой версии нейронной сети «Discernmentor»

Модель на Python представлена на рисунке 13. Метод Flatten преобразует изображение поступающее на вход в вектор.

```
model = keras.Sequential([
  Flatten(input_shape=(28, 28, 1)),
  Dense(128, activation = 'relu'),
  Dense(10, activation='softmax')
])
```

Рисунок 13 – Модель первой версии нейронной сети «Discernmentor» на Python

Функции активации использовались relu и softmax.

За 5 эпох обучения функция потерь стала равна 0,08, а точность определения составила 97,3%. На рисунке 10 приведены примеры распознанных изображений.

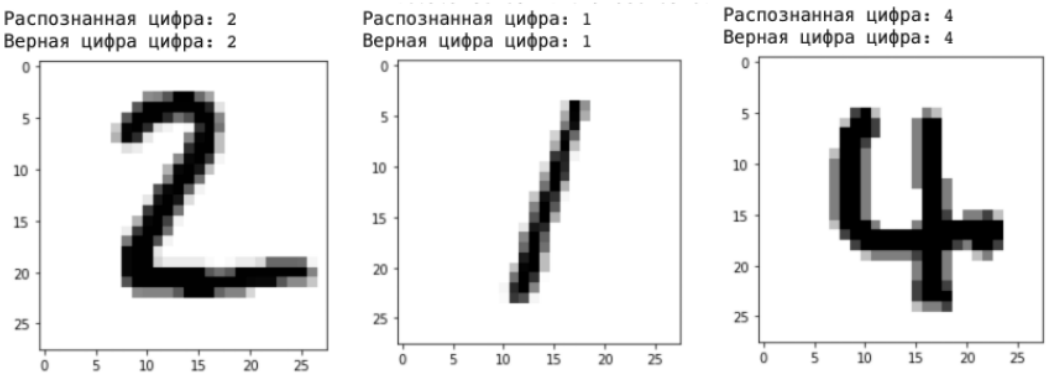

Рисунок 14 – Примеры распознанных цифр нейросети «Discernmentor» первой версии

На тестовой выборке нейронная сеть неправильно определила 248 изображений из 10 000. На рисунке 11 приведены примеры этих изображений.

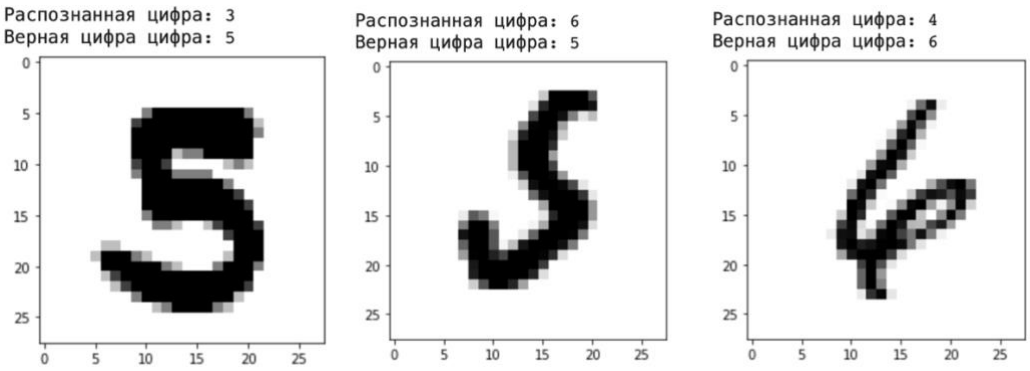

Рисунок 15 – Примеры не распознанных цифр нейросети «Discernmentor» первой версии

Как мы можем видеть, некоторые цифры не были распознаны корректно. Полный код программы “Discernmentor” версии 1 представлен в приложении Б.

3.2.2 Вторая версия

Рассмотрим вторую версию программы Discernmentor.Для обучения так же была взята выборка MNIST и была разделена на обучающую и валидационную выборки в соотношении 80/20.

После обучения нейронной сети точность определения рукописной цифры на обучающей выборке составляла 99.52%, а на тестовой 99,06%.

Результаты работы данной версии программы выше за счет другой модели нейронной сети.

В программе Discernmentor версии 2 использовалась сверточная модель нейронной сети в отличие от Discernmentor версии 1. Структура нейронной сети состоит первого сверточного слоя с 32 фильтрами, за которым выполняется операция MaxPooling, чтобы укрупнить масштаб признаков. После этих действий размерность изображения снижается до 14*14 пикселей. Затем идет сверточный слой с 64 фильтрами и операция MaxPooling, после них размерность изображения снижается до 7*7 пикселей. Полученное изображение переводится в тензер и подается на вход полносвязной

нейронной сети, состоящей из 128 нейронов. Активационная функция данного слоя – relu Последний, выходной слой состоит из 10 нейронов и имеет функцию активации softmax. Структура модели изображена на рисунке 16.

Рисунок 16 – Схема нейронной сети «Discernmentor» второй версии

Структура нейронной сети на Python показана на рисунке 17.

```
model = keras.Sequential([
    Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2), strides=2),
    Conv2D(64, (3,3), padding='same', activation='relu'),
    MaxPooling2D((2, 2), strides=2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10,  activation='softmax')
])
```

Рисунок 17 – Структура нейронной сети на Python

Результаты работы второй версии программы лучше, на тестовой выборке нейронная сеть неправильно определила всего 94 изображения из 10 000. За 5 эпох обучения функция потерь стала равна 0,03, а точность определения составила 99,02%, что значительно выше результатов первой версии нейронной сети. На рисунке 18 представлены изображения, которые не были распознаны первой версией программы.

Рисунок 18 – Результат классификации нейросети «Discernmentor» второй версии

Как мы можем видеть, некоторые из цифр, не распознанных первой версией, были распознаны второй версией программы. На практике было доказано, что сверточная с классификацией изображений нейронная сеть справляется лучше, чем полносвязная нейросеть. Полный код программы Discernmentor версии 2 представлен в приложении В.

3.3 Нейронная сеть «Creator»

Нейронная сеть была создана с генеративно-состязательной модели для генерации изображений рукописных цифр. Для этого мы обучили модель, используя набор данных MNIST, состоящий из рукописных цифр.

В обучающей выборке есть цифры с разными почерками. По мере того, как GAN изучает распределение данных, она также генерирует цифры с разными стилями рукописного ввода.

Выходные коэффициенты должны находиться в интервале от -1 до 1. Поэтому на выходе генератора мы используем гиперболическую функцию активации Tanh().

Дискриминатор на вход получает вектор из 784 элементов. Затем последовательно идут 3 скрытых слоя с 1084, 512 и 256 нейронами

соответственно. Выходной слой имеет 1 нейрон и функцию активации Sigmoid характерную для представления вероятности. На рисунке 19 представлена схема реализации дискриминатора.

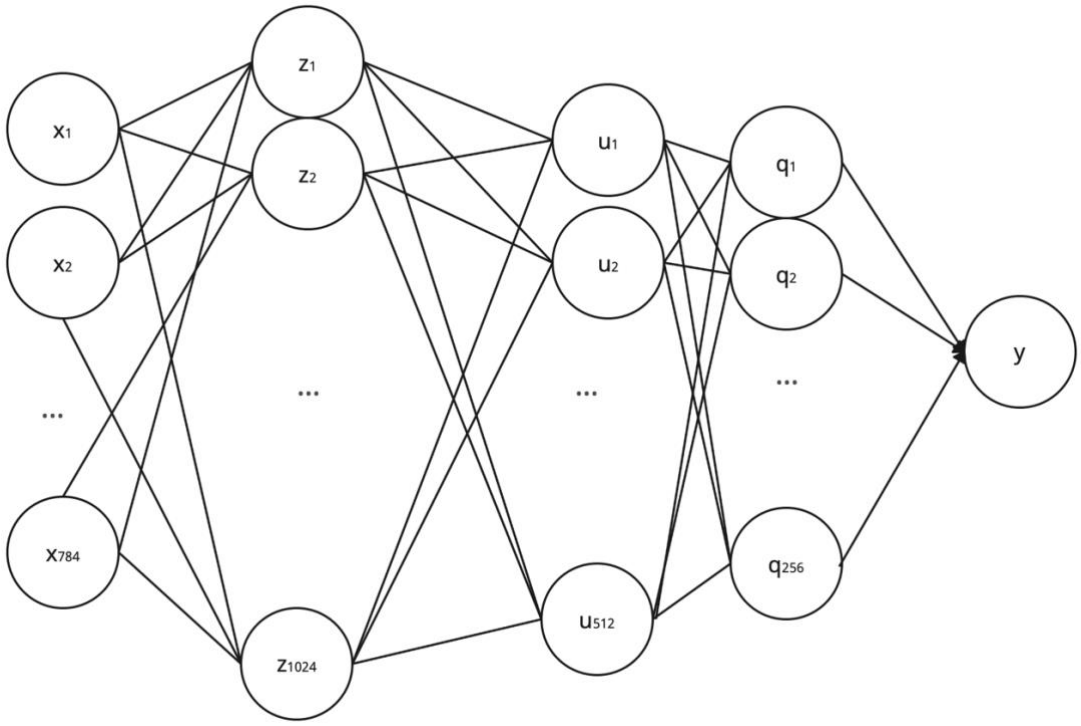

Рисунок 19 – Схема реализации Дискриминатора

В генеративно-состязательных сетях генератор – это модель, которая берет в качестве входных данных некоторую выборку из пространства скрытых переменных, напоминающих данные в обучающем наборе.

Генератор состоит из 100-мерного входа, 3 скрытых слоев имеющих 256, 512 и 1024 нейрона соответственно. На выходе генератора получается вектор размером 784, который далее преобразуется в изображение. На рисунке 20 представлена схема генератора.

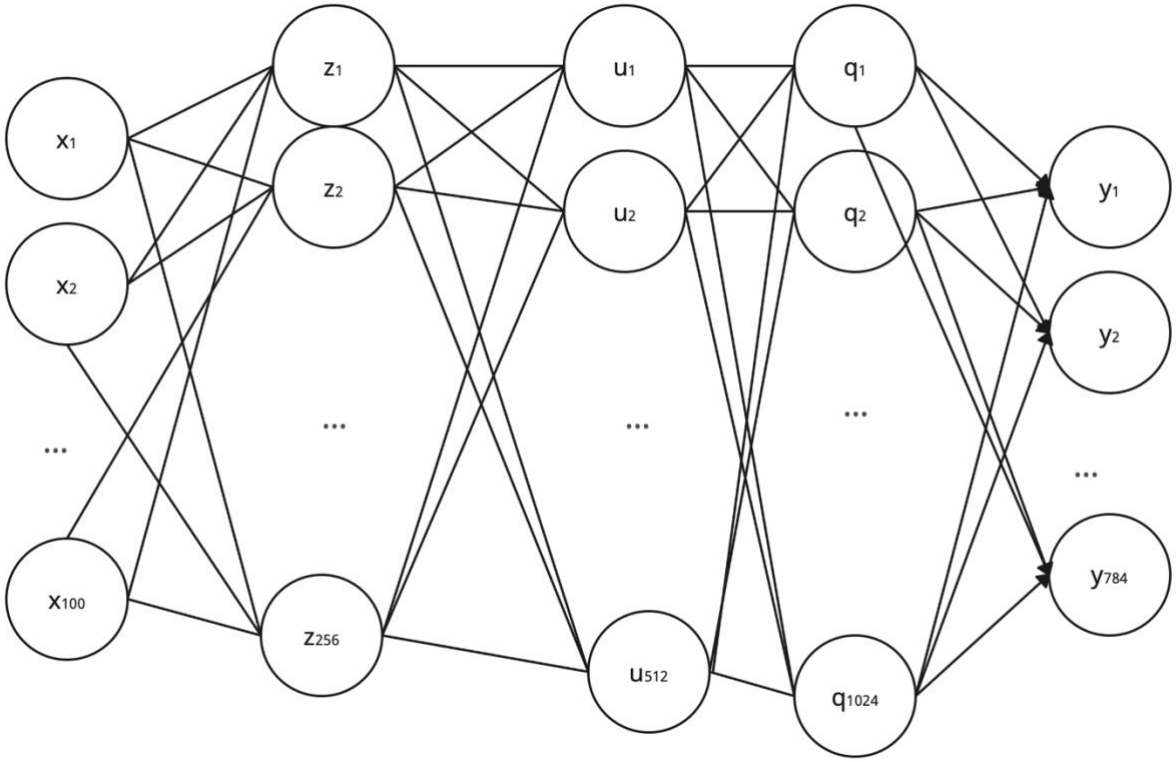

Рисунок 20 – Схема генератора в нейросети «Creator»

Теперь, когда мы определили модели для дискриминатора и генератора, мы готовы начать обучение. На рисунке 21 указаны результаты обучения дискриминатора нейронной сети, где на оси x указано количество эпох обучения, а на оси y указано значение функции потерь.

Рисунок 21 – Результаты обучения дискриминатора

На рисунке 22 изображены результаты обучения генератора, где на оси x указано количество эпох обучения, а на оси y указано значение функции потерь.

Рисунок 22 – Результаты обучения генератора

Функция потерь используется для расчета ошибки между реальными и полученными ответами. Наша глобальная цель – минимизировать эту ошибку. Таким образом, функция потерь эффективно приближает обучение нейронной сети к этой цели. В математическом плане процесс обучения GAN заключается в минимаксной игре двух игроков, в которой D адаптирован для минимизации ошибки различия реального и сгенерированного образца, а G адаптирован на максимизацию вероятности того, что D допустит ошибку.

Как мы можем видеть, результаты обучения еще можно улучшить, увеличив время обучения.

Сгенерируем несколько образцов цифр, сгенерированных нейронной сетью. Для этого передадим генератору инициирующий набор случайных чисел. На рисунке 23 представлены выходные данные работы программы.

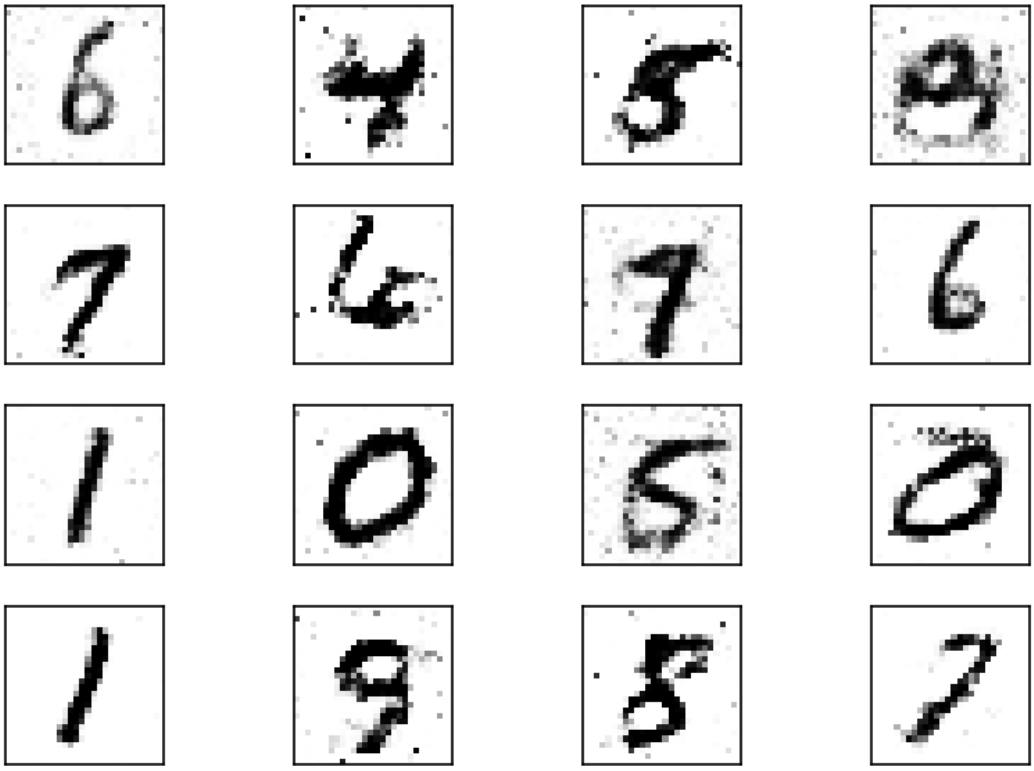

Рисунок 23 – Рукописные цифры, сгенерированные программой «Creator»

После пятидесяти эпох обучения есть несколько цифр, будто бы написанных рукой человека. Результаты можно улучшить, проводя более длительное обучение, но это займет большее количество времени, и требует больших вычислительных мощностей.

В начале процесса обучения сгенерированные изображения абсолютно случайны. По мере обучения генератор изучает распределение реальных данных, и примерно через двадцать эпох некоторые сгенерированные изображения цифр уже напоминают реальные данные. Полный код программы представлен в приложении Г.

Таким образом, нами было разработаны три нейронные сети, которые решают различные типы задач: регрессионную, классифицирующую и генерирующую с использованием Python и библиотек Keras и PyTorch.

ЗАКЛЮЧЕНИЕ

В заключение подведем основной итог курсовой работы. За время выполнения работы цель была достигнута, а именно: мы разработали три экспертные системы с помощью нейронныех сетей. Также, были выполнены все поставленные задачи:

– изучили теоретические основы нейронных сетей,

– изучили структуры нейронных сетей и правила их функционирования,

– рассмотрели основные типы задач, которые решают нейронные сети,

– провели теоретическое исследование алгоритмов обучения нейронных сетей,

– разработали нейронные сети для классификации и генерации данных с помощью библиотек Keras и PyTorch.

За время выполнения курсовой работы были рассмотрены и изучены основные технологии и принципы разработки нейронных сетей, был рассмотрен инструмент для обработки и генерации изображений.

Для разработки программного продукта было изучено такое средство разработки нейронной сети как TensorFlow, Keras и PyTorch. Были рассмотрены различные алгоритмы и структуры, используемые при работе с картинками. В целях повышения точности работы нейронных сетей был изучен алгоритм Dropout и архитектуру сверточных нейронных сетей, позволившие достичь прирост в точности работы нейронной сети.

На основе полученных знаний было разработано несколько нейронных сетей, которые делали следующие действия:

1) переводили температуру из градусов Цельсия в градусы Фаренгейта,

2) классифицировали рукописные цифры,

3) генерировали новые рукописные цифры.

В будущем планируется улучшить нейронные сети, для повышения точности и увеличения скорости обработки данных.

Конечно, на сегодняшний день ИНС способны не только различать и генерировать цифры, но и другие символы, например, рукописный текст, а также различного рода изображения, причем такие сети используют подобный алгоритм.

Данный алгоритм может быть также усовершенствован и использоваться даже для разработки различного рода программного обеспечения для персональных компьютеров.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Нейронные сети. URL: https://sbercloud.ru/ru/services/neural-networks (дата обращения 19.12.2022)
2. Сколько скрытых слоев и скрытых узлов необходимо в нейронной сети? URL: https://radioprog.ru/post/794 (дата обращения 19.12.2022)
3. Нейронные сети, перцептрон – викиконспекты. URL: https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептрон (дата обращения 19.12.2022)
4. Нейронные сети, перцептрон. URL: https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептрон (дата обращения 19.12.2022)
5. Эндрю Тарсак, Грокаем глубокое обучение / Эндрю Тарсак. СПБ.: Питер, 2021 – 352 с. – ISBN 978-5-4461-1334-7
6. Выборка – MachineLearning.ru URL: http://www.machinelearning.ru/wiki/index.php?title=Выборка (дата обращения 19.12.2022)
7. Обучение созданию нейронных сетей на Python. URL: https://hsbi.hse.ru/articles/obuchenie-sozdaniyu-neyronnykh-setey-na-python/ (дата обращения 19.12.2022)
8. Как начать работу с Keras. URL: https://timeweb.com/ru/community/articles/kak-nachat-rabotu-s-keras (дата обращения 19.12.2022)
9. Что такое PyTorch и как он работает. URL: https://www.kverner.ru/chto-takoe-pytorch-i-kak-on-rabotaet/ (дата обращения 19.12.2022)
10. PyTorch – Краткое руководство. URL: https://coderlessons.com/tutorials/python-technologies/uznaite-pytorch/pytorch-kratkoe-rukovodstvo (дата обращения 19.12.2022)

11. Google Collaboratory или Google Colab. URL: https://www.ikkaro.com/ru/Google-Collab-или-Google-Colab/ (дата обращения 19.12.2022)

12. THE MNIST DATABASE. URL: http://yann.lecun.com/exdb/mnist/ (дата обращения 19.12.2022)

13. MNIST dataset. URL: https://blog.skillfactory.ru/glossary/mnist-dataset/ (дата обращения 19.12.2022)

14. Генеративно-состязательные модели, или как сделать из нейросети художника. URL: https://smartiqa.ru/blog/neural-network-gan (дата обращения 19.12.2022)

15. Generative Adversarial Nets (GAN). URL: https://neerc.ifmo.ru/wiki/index.php?title=Generative_Adversarial_Nets_%28GAN%29 (дата обращения 19.12.2022)

16. Сигмоида. URL: https://ru.wikipedia.org/wiki/Сигмоида (дата обращения 19.12.2022)

17. Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, than. URL: https://neurohive.io/ru/osnovy-data-science/activation-functions/ (дата обращения 19.12.2022)

18. Dropout – метод борьбы с переобучением нейронной сетиURL: https://proproprogs.ru/neural_network/dropout-metod-borby-s-pereobucheniem-neyronnoy-seti (дата обращения 19.12.2022)

19. Строим градиентные алгоритмы оптимизации Adam, RMSProp, Adagrad, Adadelta. URL: https://proproprogs.ru/tensorflow/tf-stroim-gradientnye-algoritmy-optimizacii-adam-rmsprop-adagrad-adadelta (дата обращения 19.12.2022)

ПРИЛОЖЕНИЕ А

Код нейросети «Meteo»

```
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
input_data = np.array([-40, -10, 0, 8, 15, 22, 38])
output_data = np.array([-40, 14, 32, 46, 59, 72, 100])
model = keras.Sequential()
model.add(Dense(units = 1, input_shape=(1,), activation = 'linear'))
model.compile(loss = 'mean_squared_error', optimizer =
keras.optimizers.Adam(0.1), metrics=['accuracy'])
history = model.fit(input_data, output_data, epochs = 600, verbose =
0)
plt.plot(history.history['loss'])
plt.grid(True)
plt.show()
print(model.get_weights())
temperature = int(input("Введите температуру в Цельсиях:\n"))
temperature_f = float(model.predict([temperature]))
temperature_f_2 = float(temperature*1.8+32)
print(f'Температура в Фаренгейтах (предсказанная нейросетью) =
{temperature_f}')
print(f'Температура в Фаренгейтах (рассчитанная по формуле) =
{temperature_f_2}')
```

ПРИЛОЖЕНИЕ Б

Код первой версии нейросети «Discernmentor»

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Flatten
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255
x_test = x_test / 255
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
model = keras.Sequential([
  Flatten(input_shape=(28, 28, 1)),
  Dense(128, activation = 'relu'),
  Dense(10, activation='softmax')
])
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
metrics = ['accuracy'])
model.fit(x_train, y_train_cat, batch_size=32, epochs = 5,
validation_split = 0.2)
"""подаем на вход тестовую выборку"""
model.evaluate(x_test, y_test_cat)
"""Распознавание тестовой выборки"""
pred = model.predict(x_test)
pred = np.argmax(pred, axis=1)
print(pred.shape)
print(pred[:20])
print(y_test[:20])
"""Выделение неверных результатов"""
mask = pred == y_test
print(mask[:10])
```

```
x_false = x_test[~mask]
p_false = pred[~mask]
print(x_false.shape)
for i in range (20):
  print("Значение цифры = "+str(p_false[i]))
  plt.imshow(x_false[i],cmap=plt.cm.binary)
  plt.show()
```

ПРИЛОЖЕНИЕ В

Код второй версии нейросети «Discernmentor»

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D,
MaxPooling2D
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# стандартизация входных данных
x_train = x_train / 255
x_test = x_test / 255
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
x_train = np.expand_dims(x_train, axis=3)
x_test = np.expand_dims(x_test, axis=3)
print( x_train.shape )
model = keras.Sequential([
    Conv2D(32, (3,3), padding='same', activation='relu',
input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2), strides=2), # укрупняет масштаб полученных
признков
    Conv2D(64, (3,3), padding='same', activation='relu'),
    MaxPooling2D((2, 2), strides=2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10,  activation='softmax')
])
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
his = model.fit(x_train, y_train_cat, batch_size=32, epochs=5,
validation_split=0.2)
```

```
model.evaluate(x_test, y_test_cat)
pred = model.predict(x_test)
pred = np.argmax(pred, axis=1)
mask = pred == y_test
print(mask[:10])
x_false = x_test[~mask]
p_false = pred[~mask]
print(x_false.shape)
```

ПРИЛОЖЕНИЕ Г

Код нейросети «Creator»

```
import torch
from torch import nn
import math
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms
device = ""
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
torch.manual_seed(111)
transform = transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.5,), (0.5,))])
train_set = torchvision.datasets.MNIST(
    root=".", train=True, download=True, transform=transform)
batch_size = 32
train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True)
real_samples, mnist_labels = next(iter(train_loader))
for i in range(32):
    ax = plt.subplot(4, 8, i + 1)
    plt.imshow(real_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 1024),
```

```
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(x.size(0), 784)
        output = self.model(x)
        return output
discriminator = Discriminator().to(device=device)
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh(),
        )

    def forward(self, x):
        output = self.model(x)
        output = output.view(x.size(0), 1, 28, 28)
```

```
        return output
generator = Generator().to(device=device)
lr = 0.0001
num_epochs = 50
loss_function = nn.BCELoss()
optimizer_discriminator = torch.optim.Adam(discriminator.parameters(),
lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)
for epoch in range(num_epochs):
    for n, (real_samples, mnist_labels) in enumerate(train_loader):
        # Данные для тренировки дискриминатора
        real_samples = real_samples.to(device=device)
        real_samples_labels = torch.ones((batch_size, 1)).to(
            device=device)
        latent_space_samples = torch.randn((batch_size, 100)).to(
            device=device)
        generated_samples = generator(latent_space_samples)
        generated_samples_labels = torch.zeros((batch_size,
1)).to(device=device)
        all_samples = torch.cat((real_samples, generated_samples))
        all_samples_labels = torch.cat(
            (real_samples_labels, generated_samples_labels))
        # Обучение дискриминатора
        discriminator.zero_grad()
        output_discriminator = discriminator(all_samples)
        loss_discriminator = loss_function(
            output_discriminator, all_samples_labels)
        loss_discriminator.backward()
        optimizer_discriminator.step()
        # Данные для обучения генератора
        latent_space_samples = torch.randn((batch_size, 100)).to(
            device=device)
        # Обучение генератора
        generator.zero_grad()
```

```
        generated_samples = generator(latent_space_samples)
        output_discriminator_generated =
discriminator(generated_samples)
        loss_generator = loss_function(
            output_discriminator_generated, real_samples_labels)
        loss_generator.backward()
        optimizer_generator.step()
latent_space_samples = torch.randn(batch_size, 100).to(device=device)
generated_samples = generator(latent_space_samples)
generated_samples = generated_samples.cpu().detach()
for i in range(32):
    ax = plt.subplot(4, 8, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
generated_samples = generated_samples.cpu().detach()
for i in range(32):
    ax = plt.subplot(4, 8, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
generated_samples = generated_samples.cpu().detach()
for i in range(32):
    ax = plt.subplot(4, 8, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
generated_samples = generated_samples.cpu().detach()

for i in range(32):
    ax = plt.subplot(4, 8, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```