

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
Теорическая часть	6
История реализации генетического алгоритма.....	6
Виды генетического алгоритма.....	8
Степень научной разработанности.....	14
Практическая часть.....	16
Введение	16
Реализация.....	16
СПИСОК ИСТОЧНИКОВ	22

ВВЕДЕНИЕ

В последнее время все больше набирает популярность новые алгоритмы, такие как нейронные сети и генетический алгоритм. Генетический алгоритм – эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искоемых параметров с использованием механизмов, аналогичных естественному отбору в природе.

Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции (наследование, мутации, отбор и кроссинговер). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе [5].

Генетические алгоритмы предназначены для решения задач оптимизации. Примером подобной задачи может служить обучение нейросети, то есть подбора таких значений весов, при которых достигается минимальная ошибка. При этом в основе генетического алгоритма лежит метод случайного поиска. Основным недостатком случайного поиска является то, что нам неизвестно сколько понадобится времени для решения задачи.

Впервые подобный алгоритм был предложен в 1975 году Джоном Холландом (John Holland) в Мичиганском университете. Он получил название «репродуктивный план Холланда» и лег в основу практически всех вариантов генетических алгоритмов.

Любой организм может быть представлен своим фенотипом, который фактически определяет, чем является объект в реальном мире, и генотипом, который содержит всю информацию об объекте на уровне хромосомного набора. При этом каждый ген, то есть элемент информации генотипа, имеет свое отражение в фенотипе.

Таким образом, для решения задач необходимо представить каждый признак объекта в форме, подходящей для использования в генетическом алгоритме. Все дальнейшее функционирование механизмов генетического алгоритма производится на уровне генотипа, позволяя обойтись без информации о внутренней структуре объекта, что и обуславливает его широкое применение в самых разных задачах.

В наиболее часто встречающейся разновидности генетического алгоритма для представления генотипа объекта применяются битовые строки. При этом каждому атрибуту объекта в фенотипе соответствует один ген в генотипе объекта. Ген — битовая строка, чаще всего фиксированной длины, которая представляет собой значение этого признака [3].

Целью данной работы является рассмотрение реализации генетического алгоритма.

Теорическая часть

История реализации генетического алгоритма

Первые работы по имитационному моделированию эволюции были проведены в 1954 году Нильсом Баричелли на компьютере, установленном в Институте перспективных исследований Принстонского университета, опубликованная в том же году работа привлекла широкое внимание. С 1957 года австралийский генетик Алекс Фразер опубликовал серию работ по имитационному моделированию искусственного отбора среди организмов с множественным контролем измеримых характеристик.

Положенное начало позволило реализовать имитационные модели эволюционных процессов по методам, описанным в книгах Фразера и Барнелла(1970) и Кросби (1973). Имитационные модели Фразера включали все важнейшие элементы современных генетических алгоритмов. Вдобавок к этому, Ганс-Иоахим Бремерманн в 1960-х годах опубликовал серию работ, которые также принимали подход использования популяции решений, подвергаемой рекомбинации, мутации и отбору, в проблемах оптимизации. Исследования Бремерманна также включали элементы современных генетических алгоритмов. Среди прочих ранних исследователей — Ричард Фридберг, Джордж Фридман и Майкл Конрад. Множество ранних работ были переизданы Давидом Фогелем (1998).

Хотя Баричелли в работе 1963 года имитировал способности машины играть в простую игру, искусственная эволюция стала общепризнанным методом оптимизации после работы Инго Рехенберга и Ханса-Пауля Швифеля в 1960-х и начале 1970-х годов двадцатого века — группа Рехенберга смогла решить сложные инженерные проблемы согласно стратегиям эволюции. Другим подходом была техника эволюционного программирования Лоренса Дж. Фогеля, которая была предложена для создания искусственного интеллекта. Эволюционное программирование первоначально использовало

конечные автоматы для предсказания обстоятельств и разнообразие и отбор для оптимизации логики предсказания. Генетические алгоритмы стали особенно популярны благодаря работе Джона Холланда в начале 70-х годов и его книге «Адаптация в естественных и искусственных системах» (1975). Исследование Фогеля основывалось на экспериментах Холланда с клеточными автоматами и его (Холланда) трудах, написанных в университете Мичигана. Холланд ввел формализованный подход для предсказания качества следующего поколения, известный как Теорема схем. Исследования в области генетических алгоритмов оставались в основном теоретическими до середины 80-х годов, когда была, наконец, проведена Первая международная конференция по генетическим алгоритмам в Питтсбурге, Пенсильвания (США).

С ростом исследовательского интереса существенно выросла и вычислительная мощность настольных компьютеров, это позволило использовать новую вычислительную технику на практике. В конце 80-х, компания General Electric начала продажу первого в мире продукта, работавшего с использованием генетического алгоритма. Им стал набор промышленных вычислительных средств. В 1989, другая компания Axcelis, Inc. выпустила Evolver — первый в мире коммерческий продукт на генетическом алгоритме для настольных компьютеров. Журналист The New York Times в технологической сфере Джон Маркофф писал об Evolver в 1990 году [8].

Виды генетического алгоритма

Генетический алгоритм — это в первую очередь эволюционный алгоритм, другими словами, основная фишка алгоритма — скрещивание (комбинирование). Как несложно догадаться идея алгоритма взята у природы. Путем перебора и самое главное отбора получается правильная «комбинация». Алгоритм делится на три этапа: скрещивание, селекция (отбор), формирования нового поколения.

Если результат нас не устраивает, эти шаги повторяются до тех пор, пока результат нас не начнет удовлетворять или произойдет одно из нижеперечисленных условий: количество поколений (циклов) достигнет заранее выбранного максимума; исчерпано время на мутацию.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения [7].

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо фенотип, им описываемый, решает поставленную задачу.

При выборе «функции приспособленности» важно следить, чтобы её «рельеф» был «гладким».

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» —

crossover и «мутация» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Кратко реализацию можно представить в виде последовательных шагов:

1. Создание новой популяции. На этом шаге создается начальная популяция, которая, вполне возможно, окажется не подходящей, однако велика вероятность, что алгоритм эту проблему исправит. Главное, чтобы они соответствовали «формату» и были «приспособлены к размножению».

2. Размножение. Для получения потомка требуется два родителя. Главное, чтобы потомок (ребенок) мог унаследовать у родителей их черты. При это размножаются все, а не только выжившие, в противном случае выделится один альфа самец, гены которого перекроют всех остальных, а в реализации это принципиально не приемлемо.

3. Мутации. Мутации схожи с размножением, из мутантов выбирают некое количество особей и изменяют их в соответствии с заранее определенными операциями.

4. Отбор. Начинается выбор из популяции долю тех, кто «пойдет дальше». При этом долю «выживших» после отбора необходимо определить заранее руками, указывая в виде параметра. Остальные особи должны погибнуть [2].

Существуют различные модели генетического алгоритма (классический, простой генетический алгоритм, гибридный, СНС генетический алгоритм и др.). Они различаются по стратегиям отбора и формирования нового поколения особей, операторами генетического алгоритма, кодированием генов и т.д.

СНС-алгоритм

CHC (Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation) был предложен Эсхелманом и характеризуется следующими параметрами:

1. Для нового поколения выбираются N лучших различных особей среди родителей и детей. Дублирование строк не допускается.
2. Для скрещивания выбирается случайная пара, но не допускается, чтобы между родителями было мало хэммингово расстояние или мало расстояние между крайними различающимися битами.
3. Для скрещивания используется разновидность однородного кроссовера HUX (Half Uniform Crossover): ребенку переходит ровно половина битов каждого родителя.
4. Размер популяции небольшой, около 50 особей. Этим оправдано использование однородного кроссовера.

CHC противопоставляет агрессивный отбор агрессивному кроссоверу, однако все равно малый размер популяции быстро приводит ее к состоянию, когда создаются только более или менее одинаковые строки. В таком случае CHC применяет cataclysmic mutation: все строки, кроме самой приспособленной, подвергаются сильной мутации (изменяется около трети битов). Таким образом, алгоритм перезапускается и далее продолжает работу, применяя только кроссовер.

Genitor

Этот алгоритм был создан Д. Уитли. Genitor-подобные алгоритмы отличаются от классического ГА следующими тремя свойствами:

1. На каждом шаге только одна пара случайных родителей создает только одного ребенка.
2. Этот ребенок заменяет не родителя, а одну из худших особей популяции (в первоначальном Genitor – самую худшую).
3. Отбор особи для замены производится по ее рейтингу, а не по приспособленности.

В Genitor поиск гиперплоскостей происходит лучше, а сходимость быстрее, чем у классического генетического алгоритма, предложенного Холландом.

Гибридные алгоритмы

Идея гибридных алгоритмов (hybrid algorithms) заключается в сочетании генетического алгоритма с некоторым другим методом поиска, подходящим в данной задаче. В каждом поколении каждый полученный потомок оптимизируется этим методом, после чего производятся обычные для генетического алгоритма действия.

Такой вид развития называется ламарковой эволюцией, при которой особь способна обучаться, а затем полученные навыки записывать в собственный генотип, чтобы потом передать их потомкам. И хотя такой метод ухудшает способность алгоритма искать решение с помощью отбора гиперплоскостей, однако на практике гибридные алгоритмы оказываются очень удачными. Это связано с тем, что обычно велика вероятность того, что одна из особей попадет в область глобального максимума и после оптимизации окажется решением задачи.

Параллельные генетические алгоритмы

Генетические алгоритмы можно организовать как несколько параллельно выполняющихся процессов, это увеличит их производительность.

Рассмотрим переход от классического генетического алгоритма к параллельному. Для этого будем использовать турнирный отбор. Заведем $N / 2$ процесса (здесь и далее процесс подразумевается, как некоторая машина, процессор, который может работать независимо). Каждый из них будет выбирать случайно из популяции 4 особи, проводить 2 турнира и скрещивать победителей. Полученные дети будут записываться в новое поколение. Таким

образом, за один цикл работы одного процесса будет сменяться целое поколение.

Островная модель

Островная модель – это тоже модель параллельного генетического алгоритма. Она заключается в следующем: пусть у нас есть 16 процессов и 1600 особей. Разобьем их на 16 подпопуляций по 100 особей. Каждая из них будет развиваться отдельно с помощью некоего генетического алгоритма. Таким образом, можно сказать, что мы расселили особи по 16-ти изолированным островам.

Изредка (например, каждые 5 поколений) процессы (или острова) будут обмениваться несколькими хорошими особями. Этот процесс называется миграцией. Миграция позволяет островам обмениваться генетическим материалом.

Так как населенность островов обычно бывает невелика, подпопуляции будут склонны к преждевременной сходимости. Поэтому важно правильно установить частоту миграции. Чересчур частая миграция (или миграция слишком большого числа особей) приведет к смешению всех подпопуляций, и тогда островная модель будет несильно отличаться от обычного генетического алгоритма. Если же миграция будет слишком редкой, то она не сможет предотвратить преждевременного схождения подпопуляций.

Генетические алгоритмы стохастичны, поэтому при разных запусках популяция может сходиться к разным решениям (хотя все они в некоторой степени «хорошие»). Островная модель позволяет запустить алгоритм сразу несколько раз и пытаться совмещать «достижения» разных островов для получения в одной из подпопуляций наилучшего решения.

Ячеистые генетические алгоритмы

Ячеистые генетические алгоритмы (Cellular Genetic Algorithms) – модель параллельных генетических алгоритмов. Пусть дано 2500 процессов, расположенных на сетке размером 50×50 ячеек, замкнутой, как показано на

рис. 18 (левая сторона замыкается с правой, верхняя – с нижней, получается тор).

Каждый процесс может взаимодействовать только с четырьмя своими соседями (сверху, снизу, слева, справа). В каждой ячейке находится ровно одна особь. Каждый процесс будет выбирать лучшую особь среди своих соседей, скрещивать с ней особь из своей ячейки и одного полученного ребенка помещать в свою ячейку вместо родителя.

По мере работы такого алгоритма возникают эффекты, похожие на островную модель. Сначала все особи имеют случайную приспособленность (на рисунке она определяется по цвету). Спустя несколько поколений образуются небольшие области похожих особей с близкой приспособленностью. По мере работы алгоритма эти области растут и конкурируют между собой [10].

Генетические алгоритмы применяются для решения следующих задач:

1. Оптимизация функций.
2. Оптимизация запросов в базах данных.
3. Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний).
4. Настройка и обучение искусственной нейронной сети.
5. Задачи компоновки.
6. Составление расписаний.
7. Игровые стратегии.
8. Теория приближений.
9. Искусственная жизнь.
10. Биоинформатика (фолдинг белков).
11. Синтез конечных автоматов.
12. Настройка ПИД регуляторов [9].

Степень научной разработанности

С 90-х годов XX века при разработке интеллектуальных систем для решения различных прикладных задач прослеживается тенденция использования гибридных моделей, представляющих собой совокупность методов имитации интеллектуальной деятельности человека, таких как нейронные сети, экспертные системы, эвристические алгоритмы и нечёткие системы. В процессе обучения нечетких моделей ряд авторов, помимо традиционных методов оптимизации, таких, как итерационные алгоритмы, используют генетический алгоритм.

Экспериментальные исследования применения генетических алгоритмов показывают, что использование такого подхода позволяет добиться приемлемых результатов по повышению скорости и точности решаемых задач ввиду высокой скорости работы генетического алгоритма при использовании больших объёмов данных [1].

На сегодняшний день огромное количество ученых в России и в мире занимается реализацией генетического алгоритма. Большое количество работ посвящено разработке и модернизации подходом к реализации.

В настоящее время наиболее распространенным является подход по применению технологии объектно-ориентированного программирования (ООП), который позволил реализовать Генетический Алгоритм (ГА). После компиляции программный алгоритм ГА становится доступным для других разработчиков, при этом исключается потребность перекомпиляции всего исходного блока кода программы. Основная работа ведется с экземплярами таких классов - объектами. Каждый класс представляет собой шаблон, по которому создаются объекты нужного типа: родители, потомки и объекты, хранящие параметры работы алгоритма. Важной особенностью ООП - подхода является многократно используемый программный код. В основном многократное использование объекта обеспечивается его интерфейсом, т.е. его методами и свойствами, посредством которых наши объекты

взаимодействуют с окружающим миром. Построение объектов с хорошо продуманным интерфейсом, в дальнейшем при необходимости изменения его внутренней функциональности позволит добавить новый интерфейс, никоим образом не повлияв на другие объекты, которые работают с изменяемыми объектами программы [4].

Кроме этого, имеется в целом большое число скептиков, которые в принципе сомневаются в том, что целесообразно использовать генетические алгоритмы применительно к какой-либо задаче. В частности, звучат заявления, наподобие сделанного Стивеном С. Скиена, профессором кафедры вычислительной техники университета Стоуни-Брук, известного исследователя алгоритмов, лауреата премии института IEEE.

Тем не менее, несмотря на весь негатив, который периодически изливается на генетические алгоритмы и в целом целесообразность их использования – не все прислушиваются к этому, проводя собственные эксперименты и получая значимые опровержения данным утверждениям. Существует множество успешных применений генетических алгоритмов для решения задач, которые для других алгоритмов являлись сложно разрешимыми. Одним из самых ярких примеров является использование генетических алгоритмов в схемотехнике Джоном Козой, в результате которого были открыты две оригинальные схемы для построения контроллеров вывода данных.

Другим ярким примером является использование алгоритмов для оптимизации конструкции космической антенной, успешно проведенное и испытанное NASA. Кроме этих ярких примеров существует ещё множество других, что подтверждает простую мысль – генетические алгоритмы вполне имеют право на жизнь и на применение, достаточно чётко понимать, как и зачем их применять.

Практическая часть

Введение

В рамках данной практической работы я буду реализовывать нейронную сеть на языке Python 3. Структура нейронной будет взята из библиотеки Keras. Для обучения и последующей корректной работы нейронной сети будет взят уже подготовленный датасет MNIST.

MNIST (Modified National Institute of Standards and Technology) - база данных образцов рукописных цифр. База данных содержит 60.000 образцов в тренировочной выборке и 10.000 в тестовой выборке.

Реализация

Для начала необходимо импортировать библиотеки

```
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
```

Затем выгружаем образцы в переменные, где X изображение, Y целевой признак.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Изображения представлены в виде матрицы 28x28 и каждый элемент матрицы содержит в себе значение от 0 до 255, где 0 – черный, 255 – белый, значение между 0 и 255 – оттенки серого. Для того чтобы модель корректно считала данные из необходимо стандартизировать.

```
x_train = x_train / 255
x_test = x_test / 255
```

После выполнения данного кода данные будут представлены как набор вещественных чисел от 0 до 1.

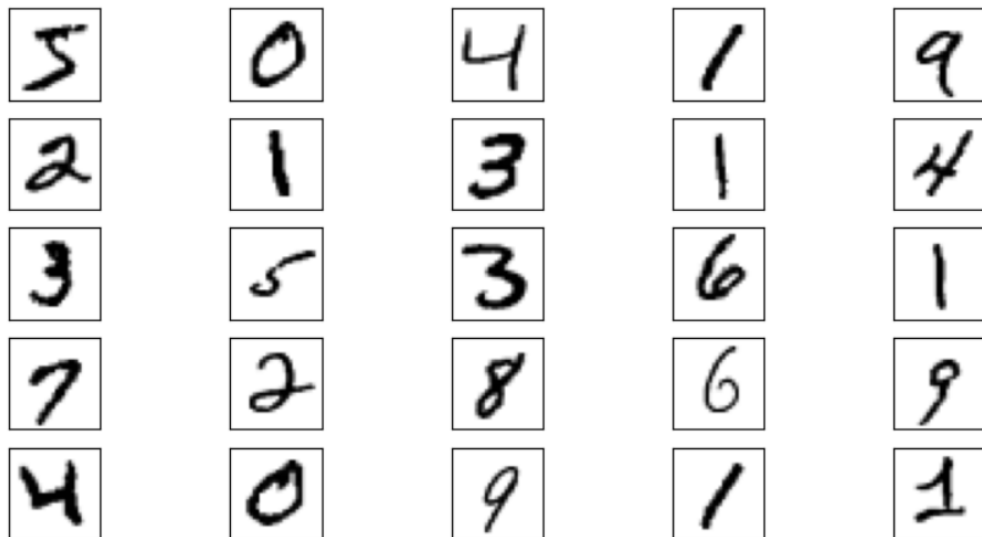
Так же изменим целевой признак, сейчас это одна цифра от 0 до 9, но я хочу представить вывод по-другому

```
y_train_cat = keras.utils.to_categorical(y_train, 10)
y_test_cat = keras.utils.to_categorical(y_test, 10)
```

После выполнения, данные представлены как массив из 10 элементов, где индекс это число, а значение вероятность отношения к классу. Например, если на изображении число два, то массив будет выглядеть следующим образом [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].

Для понимания выведем первые 25 изображений выборки.

```
plt.figure(figsize=(10,5))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap=plt.cm.binary)
plt.show()
```



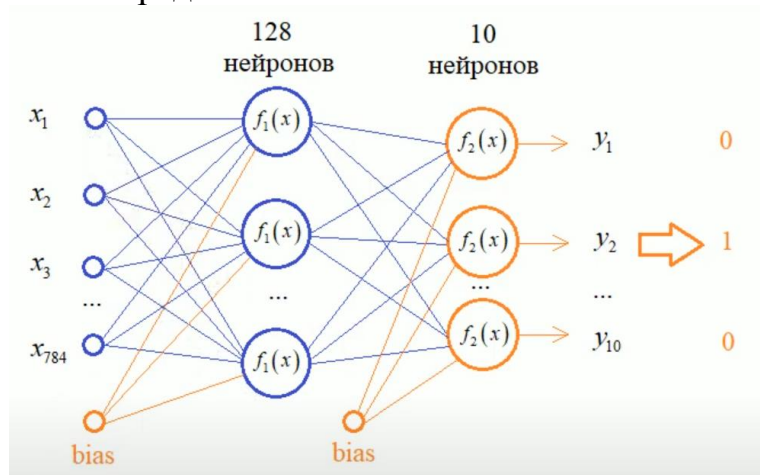
Далее создадим переменную с моделью и зададим количество и параметры слоев,

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

,где

- Flatten - это входной слой в который входит однобитная матрица 28x28 пикселей, которая преобразуется в массив данных, фактически изображенное считывается построчно.
- Dense(128) – это средний слой состоящий из 128 нейронов +1 биас скрытый нейрон.
- Dense(10) – это выходной слой

Схематически можно представить сеть так:



Выведем структуру нейронной сети

```
print(model.summary())
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290

=====
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

В Param указано количество связей между нейронами и в последствии каждой связи будет автоматически присвоен весовой коэффициент.

После того как мы определились со структурой нейронной сети её необходимо скомпилировать указав критерий качества и способ оптимизации алгоритма градиентного спуска.

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

Для данной задачи я выбрал функцию оптимизации adam и критерий качества (функция потерь) категориальная кросс энтропия. В параметре

metrics указана метрика точность, которая будет рассчитывать отношение правильных ответов к общему их количеству.

Теперь загрузить в модель тренировочный набор данных

```
model.fit(x_train, y_train_cat, # Набор тренировочных данных
          batch_size=32,
          epochs=5,
          validation_split=0.2)
```

, где

- batch_size - размер батча, то есть после 32 значений будет перерасчет коэффициентов
- epochs – количество эпох, то есть сколько раз тренировочная выборка пройдет сквозь модель
- validation_split – доля данных которая будет выделена под валидационную выборку

```
Epoch 1/5
1500/1500 [=====] - 4s 2ms/step - loss: 0.2891 - accuracy: 0.9174 - val_loss: 0.1651 - val_accuracy:
0.9538
Epoch 2/5
1500/1500 [=====] - 3s 2ms/step - loss: 0.1281 - accuracy: 0.9630 - val_loss: 0.1247 - val_accuracy:
0.9644
Epoch 3/5
1500/1500 [=====] - 3s 2ms/step - loss: 0.0878 - accuracy: 0.9741 - val_loss: 0.0985 - val_accuracy:
0.9713
Epoch 4/5
1500/1500 [=====] - 3s 2ms/step - loss: 0.0671 - accuracy: 0.9803 - val_loss: 0.0936 - val_accuracy:
0.9732
Epoch 5/5
1500/1500 [=====] - 3s 2ms/step - loss: 0.0513 - accuracy: 0.9843 - val_loss: 0.0894 - val_accuracy:
0.9738
```

После 5 циклов обучения удалось добиться точности в 0.974, соответственно 97% поданных на вход изображений распознано верно.

Теперь проверим обученную модель на тестовой выборке.

```
model.evaluate(x_test, y_test_cat)
```

```
313/313 [=====] - 1s 1ms/step - loss: 0.0831 - accuracy: 0.9735
[0.08308594673871994, 0.9735000133514404]
```

После всей проделанной работы удалось добиться точности в 97% и на тестовой выборке в том числе.

Рассмотрим на конкретном примере результаты работы. Код ниже выведет на экран изображение которое на вход получила нейронная сеть, вектор где элемент это вероятность отношения к классу и индекс с максимальным значением равный определенному нейронной сетью цифрой.

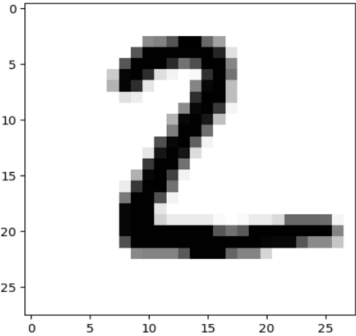
```
n = 1
x = np.expand_dims(x_test[n], axis=0)
```

```

res = model.predict(x)
print( res )
print( np.argmax(res))

plt.imshow(x_test[n], cmap=plt.cm.binary)
plt.show()
1/1 [=====] - 0s 19ms/step
[[1.0773314e-08 1.2943986e-05 9.9992335e-01 5.4808101e-05 3.9007988e-14
 4.0420241e-07 5.7024345e-09 7.0087274e-12 8.4333005e-06 1.7262469e-11]]
2
0
5
10
15
20
25
0 5 10 15 20 25

```



Для более полного понимания рассмотрим изображения, которые были определены не корректно.

```

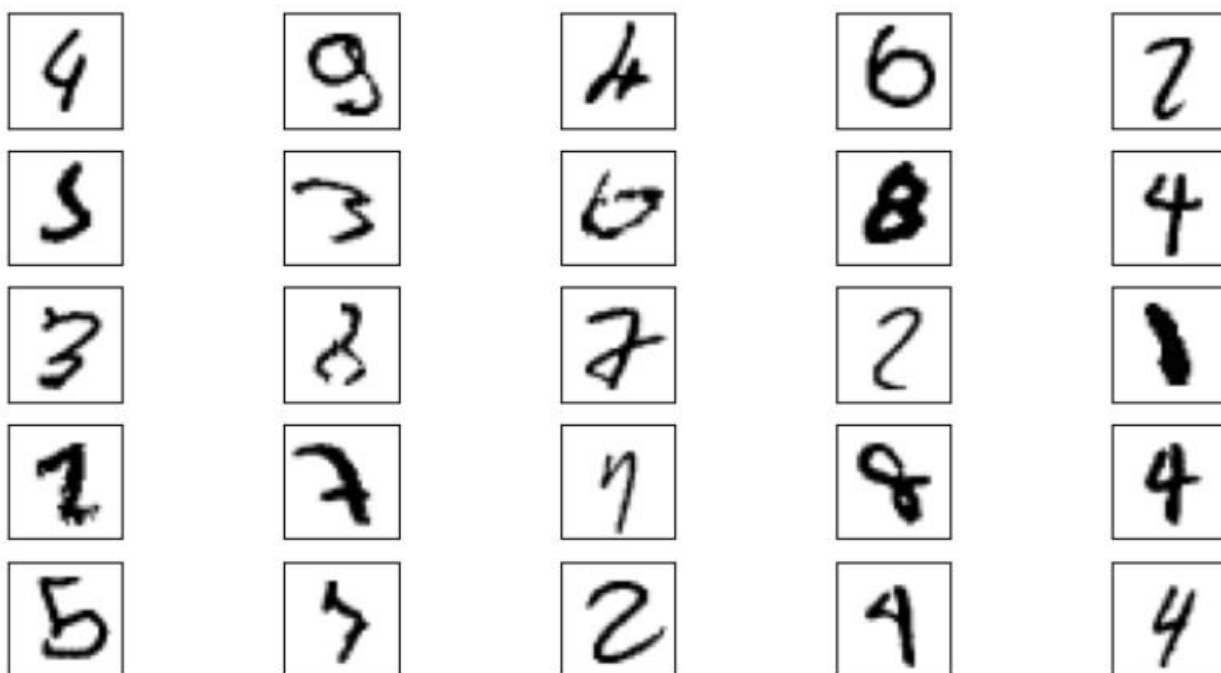
pred = model.predict(x_test)
pred = np.argmax(pred, axis=1)

mask = pred == y_test
print(mask[:10])
x_false = x_test[~mask]
y_false = x_test[~mask]

plt.figure(figsize=(10,5))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_false[i], cmap=plt.cm.binary)

plt.show()

```



Вывод:

Как видно, если цифры написаны разборчивы и на изображении нет дефектов, то нейронная сеть довольно успешно справляется с поставленной задачей, что позволяет сказать, что цели практической работы достигнуты.

В данной работе я познакомился с библиотекой Keras и представленной в ней базой данных рукописных цифр. Описал принцип написания кода и создание структуры полносвязной нейронной сети, так же описал принцип работы данной нейронной сети.

СПИСОК ИСТОЧНИКОВ

1. Свидетельство о государственной регистрации программы для ЭВМ № 2020610873 Российская Федерация. Программная реализация гибридного алгоритма размещения элементов СБИС с использованием модифицированного генетического алгоритма : № 2020610223 : заявл. 09.01.2020 : опубл. 21.01.2020 / В. И. Данильченко, Е. В. Данильченко, В. М. Курейчик ; заявитель федеральное государственное автономное образовательное учреждение высшего образования «Южный федеральный университет» (Южный федеральный университет).
2. Аралбаев, Р. А. Задачи оптимизации и применение алгоритмов генетический алгоритм на практике / Р. А. Аралбаев, А. А. Тарасов // Инновации. Наука. Образование. – 2021. – № 48. – С. 1645-1653.
3. Архипов, А. Н. Реализация генетического алгоритма с использованием компонентно-ориентированного подхода / А. Н. Архипов, А. В. Панов // Передовые инновационные разработки. Перспективы и опыт использования, проблемы внедрения в производство : Сборник научных статей по итогам десятой международной научной конференции, Казань, 30 ноября 2019 года. Том Часть 2. – Казань: Общество с ограниченной ответственностью "КОНВЕРТ", 2019. – С. 70-71.
4. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. -2-е изд. - М: Физматлит, 2006. - С. 320. 2. Игнатьева М.Н. Системный подход к определению качества образования. - М.: Аспект Пресс, 2010. - 156 с.
5. Матвеев М.Г Модели и методы искусственного интеллекта. Применение в экономике / М.Г. Матвеев, А.С Свиридов, Н.А. Алейников - Москва, Финансы и статистика; ИНФРА-М, 2008 - С. 405 - 410.
6. Матюхина, Я. С. Нечеткая система управления параметрами генетического алгоритма, настраиваемая генетическим алгоритмом / Я. С. Матюхина // Решетневские чтения. – 2017. – Т. 2. – С. 219-220.

7. Ошкин, А. В. Реализация базового генетического алгоритма на языке высокого уровня JAVA 11 / А. В. Ошкин. — Текст : непосредственный // Исследования молодых ученых : материалы XLI Междунар. науч. конф. (г. Казань, июнь 2022 г.). — Казань : Молодой ученый, 2022. — С. 6-18.
8. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 с.
9. Солодовников И.В., Доронин В. А. Генетический алгоритм для поиска логических закономерностей в данных - // Информационные технологии в управлении. № 7. М.: 2005 г.
10. Тестова, И. В. Разработка генетического алгоритма с использованием островной модели / И. В. Тестова, И. С. Оришин // Информационные системы и технологии в моделировании и управлении : сборник трудов VI Международной научно-практической конференции, Ялта, 24–26 мая 2021 года / Крымский федеральный университет имени В. И. Вернадского, Гуманитарно-педагогическая академия (филиал). – Симферополь: Общество с ограниченной ответственностью «Издательство Типография «Ариал», 2021. – С. 155-159.