Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация (РК)

КАФЕДРА Системы автоматизированного проектирования (РК-6)

НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

На тему «Определение выпуклости многоугольника традиционным методом и путем создания нейронной сети»

Студенты Глаголев Тимофей Сергеевич
Забродин Илья Николаевич
Шлюков Алексей Павлович

Группа РК6-44Б

Тип задания НИРС

Студент _____ **Глаголев Т.С.**
подпись, дата *фамилия, и.о.*

Студент _____ **Забродин И.Н.**
подпись, дата *фамилия, и.о.*

Студент _____ **Шлюков А.П.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ _____
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2024 г.

Введение:

Нейронные сети являются одним из наиболее актуальных и перспективных направлений исследований в современной науке и технологиях. Они представляют собой математические модели, вдохновленные строением и работой головного мозга человека, способные обучаться на больших объемах данных и выполнять сложные задачи обработки информации. Нейронные сети нашли широкое применение в таких областях, как машинное обучение, распознавание образов, естественный язык, компьютерное зрение, автоматическое управление и др. В данной работе мы рассмотрим основные принципы работы нейронной сети, создав ее с нуля для решения задачи об определении выпуклости многоугольника на картинке. Эту же задачу мы решим и традиционным методом.

Цель работы:

Определить, является ли прямоугольник с исходного изображения выпуклым, традиционным методом. А также создать с нуля нейросеть, которая будет решать данную задачу.

Задачи:

1. Решить задачу аналитическим методом (традиционным).
2. Изучить основные концепции нейронных сетей: ознакомиться с базовыми понятиями, определить тип архитектуры сети.
3. Создать dataset для обучения и тренировки нейронной сети, решив задачу о классификации данных.
4. Создать саму нейросеть на python, проверив ее способность к обучению.
5. Протестировать ее на изображениях многоугольника.
6. Для стандартного метода также необходимо разделить не выпуклый многоугольник на выпуклые.

Нейронные сети

Что такое нейронная сеть?

Нейронная сеть — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямиком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти.

Какие бывают нейронные сети?

Все нейронные сети можно разделить на несколько основных видов: однослойные, многослойные, прямого распространения, рекуррентные.
Однослойные сети сразу же выдают результат после загрузки в них некоторого массива данных. Многослойные сети прогоняют вводную информацию через несколько промежуточных слоев и принципом своей работы больше напоминают биологическую нейронную сеть. Выходная информация получается после прохождения всех слоев, на которых происходит обработка и анализ.
Сети прямого распространения чаще всего используются для распознавания образов, классификации и кластеризации данных — они направлены в одну сторону и не умеют перенаправлять информацию обратно. Ввели данные — получили ответ.
Рекуррентные сети перенаправляют информацию туда и обратно, пока не получат конечный результат. Они используют эффект кратковременной памяти, на основании которого информация дополняется и восстанавливается. Такие сети чаще используются для прогнозирования.
Мы будем рассматривать примеры на самом базовом типе нейронных сетей — это сеть прямого распространения (СПР).

Для чего нужны нейронные сети?

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными применениями нейронных сетей является:

Классификация — распределение данных по параметрам. Например, на вход дается набор людей и нужно решить, кому из них давать кредит, а кому нет. Эту работу может сделать нейронная сеть, анализируя такую информацию как: возраст, платежеспособность, кредитная история и тд.

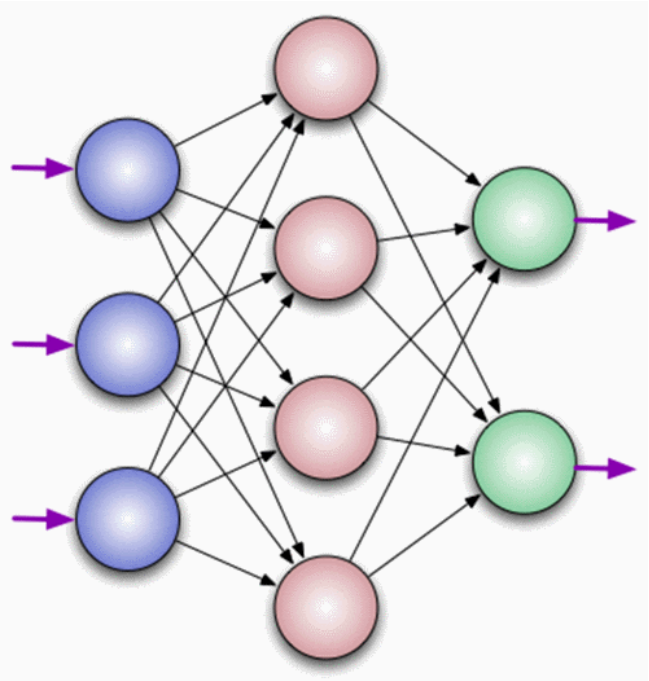
Предсказание — возможность предсказывать следующий шаг. Например, рост или падение акций, основываясь на ситуации на фондовом рынке.

Распознавание — в настоящее время, самое широкое применение нейронных сетей. Используется в Google, когда вы ищете фото или в камерах телефонов, когда оно определяет положение вашего лица и выделяет его и многое другое.

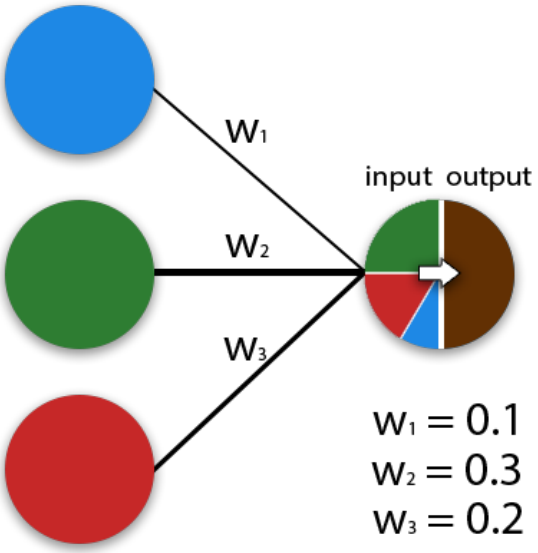
Теперь, чтобы понять, как же работают нейронные сети, давайте взглянем на ее составляющие и их параметры.

Основные понятия.

Нейрон — это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: 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. В остальных, в поле input попадает суммарная информация всех нейронов с предыдущего слоя, после чего, она нормализуется, с помощью функции активации (пока что просто представим ее f(x)) и попадает в поле output.



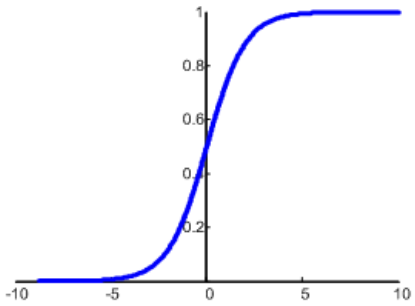
Синапс — это связь между двумя нейронами. У синапсов есть 1 параметр — вес. Благодаря ему, входная информация изменяется, когда передается от одного нейрона к другому. Допустим, есть 3 нейрона, которые передают информацию следующему. Тогда у нас есть 3 веса, соответствующие каждому из этих нейронов. У того нейрона, у которого вес будет больше, та информация и будет доминирующей в следующем нейроне (пример — смешение цветов). 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. Во время инициализации нейронной сети, веса расставляются в случайном порядке.



Важно помнить, что нейроны оперируют числами в диапазоне [0,1] или [-1,1]. Самый простой ответ — это разделить 1 на это число. Этот процесс называется нормализацией, и он очень часто используется в нейронных сетях. Для ее осуществления используют функции активации.

Функция активации — это способ нормализации входных данных (мы уже говорили об этом ранее). То есть, если на входе у вас будет большое число, пропустив его через функцию активации, вы получите выход в нужном вам диапазоне. Функций активации достаточно много поэтому мы рассмотрим самые основные: Линейная, Сигмоид (Логистическая) и Гиперболический тангенс. Главные их отличия — это диапазон значений.
Для нашего случая будет прекрасно подходить:
Сигмоид

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



Это функция активации, с диапазоном значений [0,1]. Именно на ней показано большинство примеров в сети, также ее иногда называют логистической функцией. В нашем случае не будет отрицательных значений, поэтому будет подходить именно данная функция.

Ошибка — это процентная величина, отражающая расхождение между ожидаемым и полученным ответами. Ошибка формируется каждую эпоху и должна идти на спад. Если этого не происходит, значит, вы что-то делаете не так. Ошибку можно вычислить разными путями: Mean Squared

Error (далее MSE), Root MSE и Arctan. Стоит лишь учитывать, что каждый метод считает ошибки по разному. У Arctan, ошибка, почти всегда, будет больше, так как он работает по принципу: чем больше разница, тем больше ошибка. У Root MSE будет наименьшая ошибка, поэтому используем MSE, которая сохраняет баланс в вычислении ошибки.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Нейрон смещения или bias нейрон — это третий вид нейронов, используемый в большинстве нейросетей. Особенность этого типа нейронов заключается в том, что его вход и выход всегда равняются 1 и они никогда не имеют входных синапсов. Нейроны смещения могут, либо присутствовать в нейронной сети по одному на слое, либо полностью отсутствовать. Соединения у нейронов смещения такие же, как у обычных нейронов — со всеми нейронами следующего уровня, за исключением того, что синапсов между двумя bias нейронами быть не может. Следовательно, их можно размещать на входном слое и всех скрытых слоях, но никак не на выходном слое, так как им попросту не с чем будет формировать связь.
Нейрон смещения нужен для того, чтобы иметь возможность получать выходной результат, путем сдвига графика функции активации вправо или влево.

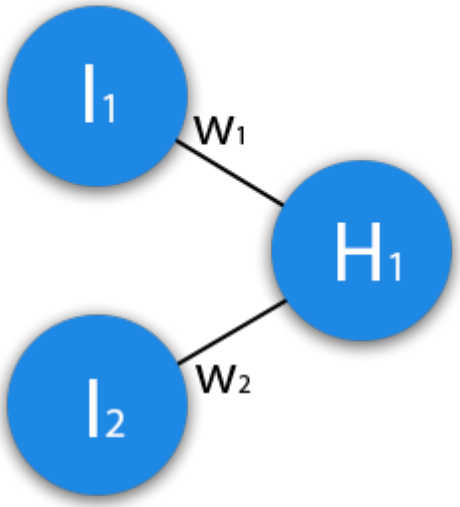
Тренировочный сет — это последовательность данных, которыми оперирует нейронная сеть.

Итерация — это своеобразный счетчик, который увеличивается каждый раз, когда нейронная сеть проходит один тренировочный сет. Другими словами, это общее количество тренировочных сетов пройденных нейронной сетью.

Эпоха. При инициализации нейронной сети эта величина устанавливается в 0 и имеет потолок, задаваемый вручную. Чем больше эпоха, тем лучше

натренирована сеть и соответственно, ее результат. Эпоха увеличивается каждый раз, когда мы проходим весь набор тренировочных сетов

Работа нейронной сети.



$$1)H_{1_{input}} = (I_1 * w_1) + (I_2 * w_2)$$

$$2)H_{1_{output}} = f_{activation}(H_{1_{input}})$$

Обучение нейронных сетей.

Существует несколько методов обучения нейронной сети (НС), основными из которых будут:

- Метод обратного распространения (Backpropagation)
- Метод упругого распространения (Resilient propagation или Rprop)
- Генетический Алгоритм (Genetic Algorithm)

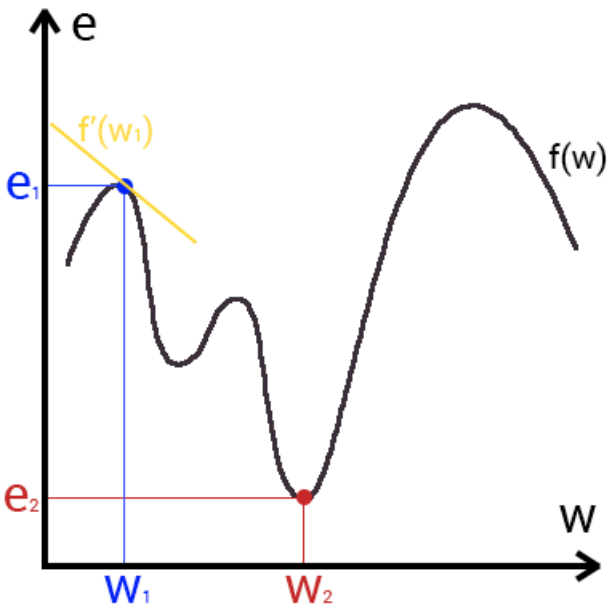
Для нашей нейросети мы будем использовать метод обратного распространения, который использует алгоритм градиентного спуска.

Переобучение

Переобучение, как следует из названия, это состояние нейросети, когда она перенасыщена данными. Это проблема возникает, если слишком долго обучать сеть на одних и тех же данных. Иными словами, сеть начнет не учиться на данных, а запоминать и “зубрить” их. Соответственно, когда вы уже будете подавать на вход этой НС новые данные, то в полученных данных может появиться шум, который будет влиять на точность результата.

Градиентный спуск.

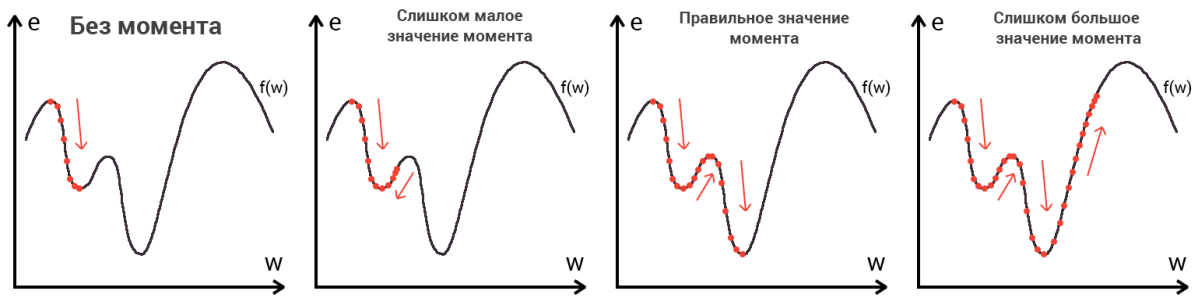
Это способ нахождения локального минимума или максимума функции с помощью движения вдоль градиента. Если вы поймете суть градиентного спуска, то у вас не должно возникнуть никаких вопросов во время использования метода обратного распространения. Для начала, давайте разберемся, что такое градиент и где он присутствует в нашей НС. Давайте построим график, где по оси x будут значения веса нейрона(w) а по оси y — ошибка соответствующая этому весу(e).



Посмотрев на этот график, мы поймем, что график функция f(w) является зависимостью ошибки от выбранного веса. На этом графике нас

интересует глобальный минимум — точка (w2,e2) или, иными словами, то место где график подходит ближе всего к оси x. Эта точка будет означать, что выбрав вес w2 мы получим самую маленькую ошибку — e2 и как следствие, самый лучший результат из всех возможных. Найти же эту точку нам поможет метод градиентного спуска (желтым на графике обозначен градиент). Соответственно у каждого веса в нейросети будет свой график и градиент и у каждого надо найти глобальный минимум.

В таком случае градиент — это вектор который определяет крутизну склона и указывает его направление относительно какой либо из точек на поверхности или графике. Чтобы найти градиент нужно взять производную от графика по данной точке (как это и показано на графике). Двигаясь по направлению этого градиента мы будем плавно скатываться в низину.
Но в большинстве случаев склон (график функции) будет волнистый и мы столкнемся с очень серьезной проблемой — локальный минимум. Попадание в локальный минимум чревато тем, что мы навсегда останемся в этой низине и никогда не скатимся с горы, следовательно мы никогда не сможем получить правильный ответ. Но мы можем избежать этого при помощи момента (momentum):



Мы можем и “проскочить” глобальный минимум, если рядом с ним есть еще низины. В конечном случае это не так важно, так как рано или поздно мы все равно вернемся обратно в глобальный минимум, но стоит помнить, что чем больше момент, тем больше будет размах с которым мы будем кататься по низинам. Вместе с моментом в методе обратного распространения также используется такой параметр как скорость обучения (learning rate). Скорость обучения, также как и момент, является

гиперпараметром — величиной, которая подбирается путем проб и ошибок.

Метод обратного распространения (МОР) или Backpropagation.

Алгоритм:

Backpropagation состоит из двух основных фаз: *forward propagation* (прямое распространение) и *backward propagation* (обратное распространение ошибки).

1. Прямое распространение:
    - На этом этапе входные данные проходят через все слои нейронной сети, вычисляя выходные значения для каждого нейрона. Каждый нейрон активируется с помощью активационной функции, такой как сигмоида.
    - Результат прямого распространения — предсказанные значения (выходы сети), которые затем сравниваются с истинными значениями (метками) для вычисления ошибки.
2. Обратное распространение ошибки:
    - На этом этапе вычисляется градиент ошибки относительно каждого веса сети, начиная с выходного слоя и двигаясь обратно к входному.
    - Используя метод градиентного спуска, веса корректируются таким образом, чтобы минимизировать ошибку. Градиенты рассчитываются с помощью производных функций активации, что позволяет определить, как небольшие изменения в весах повлияют на итоговую ошибку.

Backpropagation позволяет нейронным сетям обучаться на данных, корректируя свои внутренние параметры для улучшения точности предсказаний.

Обратное распространение ошибки состоит из нескольких этапов:

1. Вычисление ошибки на выходном слое.
2. Передача ошибки обратно через слои сети.
3. Обновление весов на основе вычисленных градиентов.

Алгоритм всегда начинается с выходного нейрона. В таком случае давайте посчитаем для него значение δ (дельта) по формуле 1.

$$1)\ \delta_O = (OUT_{ideal} - OUT_{actual}) * f'(IN)$$

$$2)\ \delta_H = f'(IN) * \sum(w_i * \delta_i)$$

Так как у выходного нейрона нет исходящих синапсов, то мы будем пользоваться первой формулой (δ output), следственно для скрытых нейронов мы уже будем брать вторую формулу (δ hidden). Тут все достаточно просто: считаем разницу между желаемым и полученным результатом и умножаем на производную функции активации от входного значения данного нейрона. Прежде чем приступить к вычислениям я хочу обратить ваше внимание на производную. Во первых как это уже наверное стало понятно, с МОР нужно использовать только те функции активации, которые могут быть дифференцированы. Во вторых чтобы не делать лишних вычислений, формулу производной можно заменить на более дружелюбную и простую формула вида:

$$f'(IN) = \begin{cases} f_{sigmoid} = (1 - OUT) * OUT \\ f_{tangh} = 1 - OUT^2 \end{cases}$$

Вся суть МОР заключается в том чтобы распространить ошибку выходных нейронов на все веса НС. Ошибку можно вычислить только на выходном уровне, как мы это уже сделали, также мы вычислили дельту в которой уже есть эта ошибка. Следственно теперь мы будем вместо ошибки использовать дельту которая будет передаваться от нейрона к нейрону.

Теперь нам нужно найти градиент для каждого исходящего синапса.

$$GRAD_B^A = \delta_B * OUT_A$$

Здесь точка A это точка в начале синапса, а точка B на конце синапса. Таким образом мы можем подсчитать градиент wi следующим образом:

Сейчас у нас есть все необходимые данные чтобы обновить вес wi и мы сделаем это благодаря функции МОР которая рассчитывает величину на которую нужно изменить тот или иной вес и выглядит она следующим образом:

$$\Delta w_i = E * GRADw + \alpha * \Delta w_{i-1}$$

Настоятельно рекомендую вам не игнорировать вторую часть выражения и использовать момент так как это вам позволит избежать проблем с локальным минимумом.

Здесь мы видим 2 константы о которых мы уже говорили, когда рассматривали алгоритм градиентного спуска: E (эпсилон) — скорость обучения, α (альфа) — момент. Переводя формулу в слова получим: изменение веса синапса равно коэффициенту скорости обучения, умноженному на градиент этого веса, прибавить момент умноженный на предыдущее изменение этого веса (на 1-ой итерации равно 0). В таком случае давайте посчитаем изменение веса wi и обновим его значение прибавив к нему Δwi.

Теперь нужно проделать данные операции для следующих нейронов, вплоть до входящих, далее - повторять это снова и снова, пока ваша ошибка не станет достаточно мала.

Процесс обучения.

Нейросеть можно обучать с учителем и без (supervised, unsupervised learning).

Обучение с учителем — это тип тренировок присущий таким проблемам как регрессия и классификация (им мы и воспользовались в примере приведенном выше). Иными словами здесь вы выступаете в роли учителя а НС в роли ученика. Вы предоставляете входные данные и желаемый результат, то есть ученик посмотрев на входные данные поймет, что нужно стремиться к тому результату который вы ему предоставили.

Обучение без учителя — этот тип обучения встречается не так часто. Здесь нет учителя, поэтому сеть не получает желаемый результат или же их количество очень мало. В основном такой вид тренировок присущ НС у которых задача состоит в группировке данных по определенным параметрам. Допустим вы подаете на вход 10000 статей на хабре и после анализа всех этих статей НС сможет распределить их по категориям основываясь, например, на часто встречающихся словах. Статьи в которых

упоминаются языки программирования, к программированию, а где такие слова как Photoshop, к дизайну.

Существует еще такой интересный метод, как обучение с подкреплением (reinforcement learning). Этот метод заслуживает отдельной статьи, но я попытаюсь вкратце описать его суть. Такой способ применим тогда, когда мы можем основываясь на результатах полученных от НС, дать ей оценку. Например мы хотим научить НС играть в PAC-MAN, тогда каждый раз когда НС будет набирать много очков мы будем ее поощрять. Иными словами мы предоставляем НС право найти любой способ достижения цели, до тех пор пока он будет давать хороший результат. Таким способом, сеть начнет понимать чего от нее хотят добиться и пытается найти наилучший способ достижения этой цели без постоянного предоставления данных “учителем”.

Также обучение можно производить тремя методами: стохастический метод (stochastic), пакетный метод (batch) и мини-пакетный метод (mini-batch). Существует очень много статей и исследований на тему того, какой из методов лучше и никто не может прийти к общему ответу. Я же сторонник стохастического метода, однако я не отрицаю тот факт, что каждый метод имеет свои плюсы и минусы.

Вкратце о каждом методе:

Стохастический (его еще иногда называют онлайн) метод работает по следующему принципу — нашел Δw, сразу обнови соответствующий вес. Мы используем именно стохастический метод для МОРа.

Пакетный метод же работает по другому. Мы суммируем Δw всех весов на текущей итерации и только потом обновляем все веса используя эту сумму. Один из самых важных плюсов такого подхода — это значительная

экономия времени на вычисление, точность же в таком случае может сильно пострадать.

Мини-пакетный метод является золотой серединой и пытается совместить в себе плюсы обоих методов. Здесь принцип таков: мы в свободном порядке распределяем веса по группам и меняем их веса на сумму Δw всех весов в той или иной группе.

Заключение исследования.

Мы ознакомились с общим устройством нейронных сетей. Также изучили основные понятия и термины, которыми будем пользоваться. Выбрали модель НС, наиболее подходящую под цель работы - СПР. А также сделали выбор метода обучения нашей НС, backpropagation. Теперь мы готовы приступать к реализации нашей НС и выполнению задач.

Разработка и реализация проектных решений.

В качестве проектного решения было предложена разработка программы для нахождения по фотографии многоугольника выпуклый он или вогнутый двумя способами: традиционным методом и путем создании нейросети.

Стандартный метод

Первый метод заключается в том, что программа предназначена для автоматической обработки изображений, на которых изображены многоугольники, с целью определения их выпуклости с помощью компьютерного зрения. Она считывает изображение, находит контуры многоугольников, аппроксимирует их до упрощенных полигонов и проверяет, являются ли они выпуклыми.

Аппроксимация контуров в нашей программе играет ключевую роль в обработке изображений, на которых изображены многоугольники.

Аппроксимация в нашем случае нужна для:

1) Упрощения формы:

   Изображения реальных объектов часто содержат излишние детали и шумы, которые могут запутать алгоритмы анализа. Аппроксимация контуров позволяет сгладить эти детали, заменяя сложные и зубчатые линии простыми и четкими многоугольниками.

2) Сокращения количества данных:

   Вместо того чтобы работать с многочисленными точками, составляющий оригинальный контур, аппроксимация позволяет работать только с нужными нам вершинами.

Благодаря аппроксимации, мы находим вершины многоугольника, с которым будем дальше работать. Проверка выпуклости осуществляется с помощью особой функции, которая вычисляет векторные произведения последовательных тройек вершин. Если все произведения одного знака, то многоугольник выпуклый.

Но вероятность работы такой программы не всегда точна, ведь компьютеру тяжело увидеть точку, которая будет находится на одной прямой между двумя другими точками, или делать настолько небольшое отклонение, которая как понятно будет видно человеческому глазу, но не видно компьютеру.

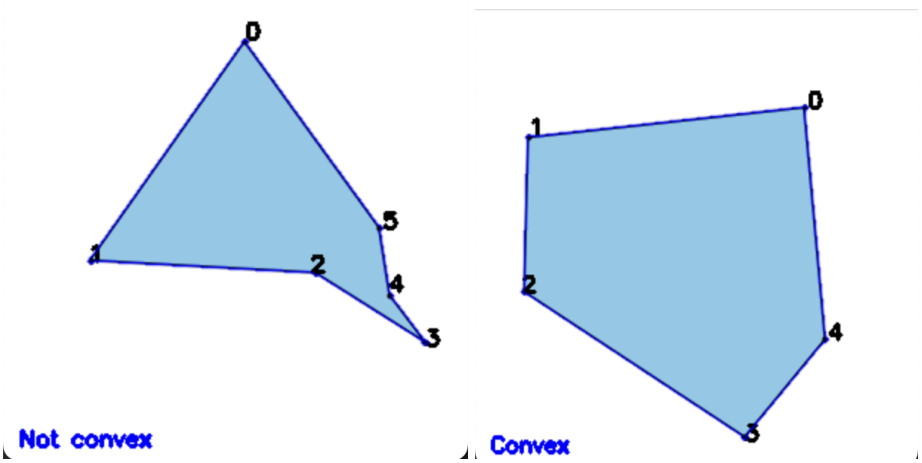
Код программы:

```
import numpy as np
import cv2 as cv
# Функция проверки, является ли многоугольник выпуклым
def is_convex_polygon(vertices):
    num_vertices = len(vertices)
    # Функция для вычисления векторного произведения трех точек
    def cross_product(p1, p2, p3):
        return (p2[0] - p1[0]) * (p3[1] - p1[1]) - (p2[1] - p1[1]) * (p3[0] - p1[0])
    sign = None
    for i in range(num_vertices):
        p1 = vertices[i]
        p2 = vertices[(i + 1) % num_vertices]
        p3 = vertices[(i + 2) % num_vertices]
        cp = cross_product(p1, p2, p3)

        if cp != 0:  # Если произведение не равно нулю
            if sign is None:
                sign = cp > 0
            elif sign != (cp > 0):
                return False  # Найдено противоположное направление, значит многоугольник не
выпуклый
    return True  # Многоугольник выпуклый
# Функция проверки, находится ли точка на достаточном расстоянии от других точек
def is_far_enough(point, vertices, min_dist=10):
    for v in vertices:
        if np.linalg.norm(np.array(point) - np.array(v)) < min_dist:
            return False
    return True
# Функция обработки изображения
def process_image(filename):
    # Чтение изображения
    img = cv.imread(filename)
    if img is None:
        print("Error: Image not found or could not be loaded.")
        return
    # Преобразование изображения в оттенки серого
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Применение пороговой бинаризации и инверсия цветов
    _, thresh = cv.threshold(gray, 127, 255, cv.THRESH_BINARY_INV)
    # Поиск контуров на изображении
    contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        print("Контуры не найдены")
        return
    vertices = []
    for cnt in contours:
        # Аппроксимация контура до многоугольника
```

```
        epsilon = 0.01 * cv.arcLength(cnt, True)  # Степень апроксимации, чем меньше, тем
полученный многоугольник к исходному
        approx = cv.approxPolyDP(cnt, epsilon, True)
        for vertex in approx:
            p = tuple(vertex.ravel())  # Преобразование координат вершины в кортеж
            if is_far_enough(p, vertices, min_dist=10):
                vertices.append(p)  # Добавление вершины в список, если она на достаточном
расстоянии
    print("Всего точек - ", len(vertices))
    if len(vertices) < 4:
        print("Найдено меньше 4 точек, что нам не подхлодит", end="\n\n")
        return
    # Проверка, является ли многоугольник выпуклым
    if is_convex_polygon(vertices):
        convex_text = "Convex"
        print("Выпуклый", end="\n\n")
    else:
        convex_text = "Not convex"
        print("Вогнутый", end="\n\n")
    # Отображение индексов вершин на изображении
    for i, p in enumerate(vertices):
        cv.putText(img, str(i), p, cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2)
    # Отрисовка контуров на изображении
    cv.drawContours(img, contours, -1, (255, 0, 0), 1, cv.LINE_AA, hierarchy, 1)
    # Добавление текста о выпуклости многоугольника внизу изображения
    text_position = (10, img.shape[0] - 10)
    cv.putText(img, convex_text, text_position, cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0),
2)
    # Отображение изображения с контурами и текстом
    cv.imshow('contours', img)
    # Ожидание нажатия клавиши для закрытия окна
    key = cv.waitKey()
    if key == 27:  # Клавиша 'Esc'
        cv.destroyAllWindows()
if __name__ == '__main__':
    for i in range(20, 31):
        filename = f"data/polygon{i:03d}.png"
        print(filename)
        process_image(filename)
```

Пример работы программы:



Для проверки корректности работы был придуман генератор многоугольников. Этот генератор предназначен для создания случайных многоугольников, которые могут быть как выпуклыми, так и вогнутыми. Он позволяет генерировать изображения с многоугольниками, которые могут быть использованы для тестирования алгоритмов обработки изображений, машинного обучения или других целей, связанных с геометрией и компьютерным зрением. Многоугольники создаются выпуклыми и вогнутыми в соотношении 1 к 1.

Реализация через нейросеть

Создание генератора изображений многоугольников

Основные логика работы генератора заключается в том, что мы задаем случайным образом количество вершин, после чего он раскидывает вершины в разные координаты, после чего сортируем их против часовой стрелки для работы с ними. Далее рисуем контуры между точками, которые потом заливаем голубым цветом, чтобы не было внутренних контуров. После чего сохраняем фотографию для дальнейшей ее обработки нашей уже созданной программой, мы сохраняем картинку и начинаем работать с ней, как в первой программе и если многоугольник нам подходит, то мы его сохраняем в папку, которая будет нашим датасетом для обучения нейросети.

Код программы:

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import random
import os
import wtf
def generate_random_polygon(num_points, i, flag, size=400, margin=25):
    """
    Генерация случайного многоугольника из заданного числа точек с заливкой.
```

```
    :param num_points: Количество точек многоугольника
    :param i: Номер текущего многоугольника для именования файла
      :param flag: Флаг для определения выпуклого или вогнутого многоугольника (1 для выпуклого, 0 для вогнутого)
    :param size: Размер изображения (по умолчанию 400x400 пикселей)
    :param margin: Отступ от краев изображения (по умолчанию 25 пикселей)
    :return: 1 если успешное создание выпуклого/вогнутого многоугольника, иначе 0
    """
    if num_points<4:
        print("Для определения выпуклости у многоугольника должно быть больше 3 точек")
        return 0

    # Создание случайных координат с учетом отступов
    points = (np.random.rand(num_points, 2) * (size - 2 * margin)) + margin

    # Найдем центр точек
    centroid = np.mean(points, axis=0)

    # Рассчитаем углы от центра для сортировки точек
    angles = np.arctan2(points[:, 1] - centroid[1], points[:, 0] - centroid[0])

    # Сортировка точек по углам против часовой стрелки
    sorted_points = points[np.argsort(angles)]

    # Построение многоугольника с заливкой
    plt.figure(figsize=(size / 100, size / 100), dpi=100)
    plt.fill(sorted_points[:, 0], sorted_points[:, 1], edgecolor='black', fill=True, facecolor='skyblue')
    plt.plot(sorted_points[:, 0], sorted_points[:, 1], 'o', color='black', markersize=0)

    # Настройка графика
    plt.xlim(0, size)
    plt.ylim(0, size)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.axis('off')

    # Сохранение изображения во временный файл
    temp_filename = 'temp_image.png'
    plt.savefig(temp_filename, bbox_inches='tight', pad_inches=0)
    plt.close()

    # Загрузка сохраненного изображения для обработки
    img = cv.imread(temp_filename)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
     _, thresh = cv.threshold(gray, 127, 255, cv.THRESH_BINARY_INV) # Применение пороговой бинаризации и инверсия цветов

    # Поиск контуров на изображении
    contours, _ = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        os.remove(temp_filename)
        return 0

    vertices = []
    for cnt in contours:
        # Аппроксимация контура до многоугольника
        epsilon = 0.01 * cv.arcLength(cnt, True)
            approx = cv.approxPolyDP(cnt, epsilon, True)      # Результат аппроксимации содержащий вершины контура

        for vertex in approx:
            p = tuple(vertex.ravel())  # Преобразование координат вершины в кортеж
            if wtf.is_far_enough(p, vertices, min_dist=15):
                vertices.append(p)  # Добавление вершины в список, если она на достаточном расстоянии

    if len(vertices) < 4:
        os.remove(temp_filename)
```

```
        return 0

    is_convex = wtf.is_convex_polygon(vertices)

    # создает выпуклые и вогнутые многоугольники в соотношении 1 к 1
    if flag == 1 and is_convex:
        # Переименование временного файла в целевую папку
        target_filename = f"data/polygon{i:04d}.png"
        os.rename(temp_filename, target_filename)
        return 1
    elif flag == 0 and not is_convex:
        target_filename = f"data/polygon{i:04d}.png"
        os.rename(temp_filename, target_filename)
        return 1

    os.remove(temp_filename)
    return 0

if name == '__main__':
    os.makedirs("data", exist_ok=True)
    cou = 1
    while cou != 2001:
        num_points = random.randint(4, 7)  # Случайное количество точек для многоугольника
        flag = cou % 2  # 1 для нечетных, 0 для четных
        if generate_random_polygon(num_points, cou, flag) == 1:
            cou += 1
```

Создание датасета

Dataset представляет собой структурированную коллекцию данных, собранную для конкретных исследовательских или обучающих целей. В рамках нашего проекта мы разработали генератор, который создает случайные изображения многоугольников, после чего эти изображения были обработаны для извлечения геометрических данных. Мы конвертировали эти данные в массивы NumPy и сохраняли их в формате .npz, создавая удобный и гибкий ресурс для последующего обучения нейронной сети.
Было принято решение взять способ хранения данных аналогично датасету mnist, в котором хранятся изображения цифр. В разделе **x** хранятся массив, соответствующие изображениям многоугольников, в массиве **y** же - “маркеры ” этих фигур (выпуклый - 1; вогнутый - 0). Данные метки мы будем получать за счет работы уже написанного стандартного метода.

Для это был написан код программы:

```
from PIL import Image
import numpy as np
import os
import nirs_test

directory = 'data/'
arr_lbs = []
arr_ims = []

for im_file in os.listdir(directory):
```

```
  f = os.path.join(directory, im_file)
  img = np.array(Image.open(f), dtype='uint8')
  mark = nirs_test.process_image(f)

  arr_ims.append(img)
  X_array = np.asarray(arr_ims)

  if not mark is None:
     arr_lbs.append(mark)
     Y_array = np.asarray(arr_lbs)
     # convex=1 convac=0

  np.savez('network_dataset.npz', x=X_array, y=Y_array)
```

Подключение датасета к НС

Теперь остается подключить созданный датасет к программе обучения нашей НС, чтобы тренировать ее на сгенерированных данных.

Код программы:

```
import numpy as np

def load_dataset():
  with np.load("network_dataset.npz", allow_pickle=True) as f:
     # convert from RGB to Unit RGB
     x_train = 1 - (f['x'].astype("float32") / 255)

     # reshape from (100, 77, 77, 4) into (100, 23761)
     x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1] * x_train.shape[2] * x_train.shape[3]))

     # labels
     y_train = f['y']

     # convert to output layer format
     y_train = np.eye(2)[y_train]

     return x_train, y_train
```

Написание алгоритма обучения НС и проверка ее работы

Далее переходим к созданию нейросети. Для ее обучения будем использовать метод обратного распространения, описанный выше. Изначально создадим матрицы весов, заполнив их случайными числами нужного диапазона. После создадим цикл, соответствующей проходу по нашим эпохам и каждую эпоху будем тренировать нашу НС (проходя сначала по всем слоям прямым распространением, а далее уже применяя МОР). Также будем подсчитывать ошибку и точность нашей НС, чтобы отслеживать ее обучение.

Обучение НС можно также сохранить в .npy файл и далее просто выгружать из него данные.

Остается только обработать поданное на вход изображение и протестировать работу нашей НС.

Код программы:

```
import numpy as np
import matplotlib.pyplot as plt

import utils

images, labels = utils.load_dataset()

weights_input_to_hidden = np.random.uniform(-0.5, 0.5, (20, 379456))
weights_hidden_to_output = np.random.uniform(-0.5, 0.5, (2, 20))
bias_input_to_hidden = np.zeros((20, 1))
bias_hidden_to_output = np.zeros((2, 1))

epochs = 5
e_loss = 0
e_correct = 0
learning_rate = 0.01

for epoch in range(epochs):
  print(f"Epoch №{epoch}")

  for image, label in zip(images, labels):
    image = np.reshape(image, (-1, 1))
    label = np.reshape(label, (-1, 1))

    # Forward propagation (to hidden layer)
    hidden_raw = bias_input_to_hidden + weights_input_to_hidden @ image
    hidden = 1 / (1 + np.exp(-hidden_raw)) # sigmoid

    # Forward propagation (to output layer)
    output_raw = bias_hidden_to_output + weights_hidden_to_output @ hidden
    output = 1 / (1 + np.exp(-output_raw))

    # Loss / Error calculation
    e_loss += 1 / len(output) * np.sum((output - label) ** 2, axis=0)
    e_correct += int(np.argmax(output) == np.argmax(label))

    # Backpropagation (output layer)
    delta_output = output - label
    weights_hidden_to_output += -learning_rate * delta_output @ np.transpose(hidden)
    bias_hidden_to_output += -learning_rate * delta_output

    # Backpropagation (hidden layer)
    delta_hidden = np.transpose(weights_hidden_to_output) @ delta_output * (hidden * (1 - hidden))
    weights_input_to_hidden += -learning_rate * delta_hidden @ np.transpose(image)
    bias_input_to_hidden += -learning_rate * delta_hidden

    # DONE

  # print some debug info between epochs
  print(f"Loss: {round((e_loss[0] / images.shape[0]) * 100, 3)}%")
  print(f"Accuracy: {round((e_correct / images.shape[0]) * 100, 3)}%")
  e_loss = 0
  e_correct = 0

# CHECK CUSTOM
test_image = plt.imread("data/polygon001.png", format="png")
plt.imshow(test_image)

test_image = 1 - (test_image.astype("float32") / 255)
```

```
# Reshape
test_image = np.reshape(test_image, (test_image.shape[0] * test_image.shape[1] * test_image.shape[2]))

# Predict
image = np.reshape(test_image, (-1, 1))

# Forward propagation (to hidden layer)
hidden_raw = bias_input_to_hidden + weights_input_to_hidden @ image
hidden = 1 / (1 + np.exp(-hidden_raw)) # sigmoid
# Forward propagation (to output layer)
output_raw = bias_hidden_to_output + weights_hidden_to_output @ hidden
output = 1 / (1 + np.exp(-output_raw))

plt.title(f"NN suggests the CUSTOM number is: {output.argmax()}")
plt.show()
```

Список литературы:

1. Gonzalez, Rafael C., and Woods, Richard E. "Digital Image Processing." Pearson Education, 2017.
2. Bradski, Gary, and Kaehler, Adrian. "Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library." O'Reilly Media, 2017.
3. Matplotlib Documentation. URL: https://matplotlib.org/stable/contents.html NumPy Documentation. URL: https://numpy.org/doc/stable/
4. OpenCV Documentation. URL: https://docs.opencv.org/master/
5. Python Software Foundation. "Python Language Reference." URL: https://docs.python.org/3/reference/index.html
6. “Наглядное введение в нейросети на примере распознавания цифр.” URL: https://proglib.io/p/neural-network-course
7. “Метод обратного распространения ошибки.” URL: https://ru.wikipedia.org/wiki/Метод_обратного_распространения_ошибки
8. “Алгоритм Backpropagation на Python” URL: https://habr.com/ru/companies/otus/articles/816667/
9. “Нейронные сети для начинающих. Часть 1/2” URL: https://habr.com/ru/articles/313216/

   https://habr.com/ru/articles/312450/
10. “Выбор слоя активации в нейронных сетях: как правильно выбрать для вашей задачи” URL: https://habr.com/ru/articles/727506/
11. Numpy manual URL: https://numpy.org/
12. “Создай свою нейросеть” Тарик Рашид URL: https://radkopeter.ru/projects/translation-neural/files/make-your-own-neural-network.pdf