Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б11-мм

Генерация нейронных сетей для моделирования поведения динамических систем

АЛИМОВ Павел Геннадьевич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
инженер-исследователь В. И. Гориховский

Санкт-Петербург
2023

Оглавление

Введение

Необходимость моделирования динамических систем возникает во многих современных областях физики [1, 14, 5, 8], экономики [2], биологии [18, 15] и математики [4, 3]. При этом, во время моделирования, часто требуется серийное нахождение решений дифференциальных уравнений (ДУ) для большого набор близких параметров системы. Но существующие численные методы нахождения решения для больших задач, на сотни и тысячи уравнений, могут оказаться очень ресурсоёмкими. Поэтому авторы из указанных выше статей для аппроксимации решений дифференциальных уравнений прибегают к методам машинного обучения, в частности к нейронным сетям (НС).

На данный момент все инструменты нахождения решений ДУ с помощью нейронных сетей разрабатываются под конкретную динамическую систему и не приспособлены для моделирования других [16, 4, 3]. Поэтому существует нужда в разработке универсального инструмента, позволяющего по форме дифференциального уравнения предсказывать параметры нейронной сети, которая лучше всего подходит для аппроксимации его решения.

Возможность применимости НС для моделирования динамических систем исследована во многих работах [16, 3, 9]. Авторы получили положительную оценку для такого подхода решения дифференциальных уравнений. Данная работа является продолжение выпускной квалификационной работы Полетанского Виктора [21], в которой была изучена применимость методов машинного обучения для аппроксимации дифференциальных уравнений широкого класса, от линейные обыкновенных до уравнений в частных производных, и посвящена разработке инструмента для генерации нейронных сетей по заданному ДУ, с дальнейшей публикацией в качестве Python пакета.

В данной работе проведён обзор научной литературы, посвящённой нахождению решений дифференциальных уравнений с помощью нейронных сетей и существующим наработкам в данной области. На основе предъявленных требований к инструменту реализован его про-

тотип с помощью выбранного фреймворка для работы с НС. На основе полученного прототипа было проведено его тестирование и создан набор данных с результатами нейронных сетей с различной топологией и параметрами обучения для выбранных в данной работе дифференциальных уравнений. Полученный набор датасетов выступит в качестве тестов для желаемого инструмента.

1 Обзор литературы и актуальность задачи

Необходимость моделирования динамических систем возникает во многих отраслях. Ниже приведены примеры работ, в которых авторы использовали нейронные сети при решении исследовательских, технических и финансовых задач.

Моделирование потока разреженного газа является одной из самых вычислительно сложных задач физической механики. В работе [1] авторам требовалось многократное нахождение решения системы дифференциальных уравнений с помощью прямого моделирования методом Монте-Карло (DSMC). Для уменьшения вычислительной сложности расчётов авторы разработали и обучили нейронную сеть, позволяющую аппроксимировать численные решения. Такая аппроксимация имеет небольшую погрешность и значительно увеличивает производительность расчета газодинамических характеристик.

Модель ценообразования опционов Блэка-Шоулза появилась в 70-х годах XX века и до сих пор активно используется на биржах для расчёта цены опционов. Harsha в своей работе [2] описывает нахождение решения дифференциального уравнения, описывающего модель Блэка-Шоулза, с помощью метода Галеркина с глубокими нейронными сетями (DGM). Сравнив аналитическое решение и полученное с помощью глубоких нейронных сетей, автор получил, что аппроксимация дала высокую точность.

Реконструкция биологических сетей с определённой функцией является сложной задачей в системной биологии. В исследовании [18] авторы проводили моделирование биологических сетей, содержащих более 10 узлов. Для решения этой задачи была реализована многослойная нейронная сеть, основанная на перцептронах. Было показано, что полученное решение может выполнять ожидаемую биологическую функцию, и был сделан вывод о применимости и эффективности подобного подхода. В другой работе [15] описана методология реализации нейронных сетей для моделирования системы обоняния у насекомых.

Математические модели ионных каналов сердца широко используются для изучения и прогнозирования поведения ионных токов, однако существуют различия между результатами моделирования и эмпирическими данными, вызванные абстрактностью состояний конформации и грубых предположениях о величине скорости переходов между ними. В работе [14] сделана попытка смягчить ограничения традиционного численного подхода к моделированию проводимости ионного канала с помощью применения нейронных сетей. В ходе работы авторы реализовали и обучили многослойные перцептронные сети для частичной и полной аппроксимации сложновычислимых правых частей дифференциальных уравнений, оценили их эффективность и показали, что НС смогли устранить недостающую динамику в системе.

Моделирование колебательной и химической кинетики многоатомных газов является задачей, в которой применение традиционных методов не представляется возможным, в связи с тем, что число дифференциальных уравнений в системе может превышать несколько тысяч. В рамках проекта "Машинное обучение в задачах неравновесной аэромеханики" в статье [5] — авторы анализируют возможность разработки комплексных решений включающих машинное обучение в качестве частей метода. Такой подход был далее применен в [8] для решения задачи моделирования колебательной кинетики углекислого газа. Это позволило впервые провести моделирование релаксации CO_2 за фронтом плоской ударной волны как в полной поуровневой постановке (STS) так и в четырехтемпературной.

Несмотря на то, что в решении множества различных задач успешно применяются нейронные сети, оказывается, что не существует универсального инструмента, позволяющего быстро пользоваться таким подходом. Текущие наработки предлагают только решения частных случаев.

В исследовании 2017-2019г. Raissi et al. были разработаны нейронные сети для аппроксимации решений небольшого набора уравнений в частных производных: Бюргера, Шрёдингера, Навье-Стокса, Аллен-Кана и Кортевега — де Фриза [16].

Flamant et al. разрабатывают универсальную нейронную сеть, которая может аппроксимировать решение любого дифференциального уравнения [7]. В качестве результата авторы получили точность сопоставимую с методом Рунге-Кутта 4 порядка при схожем количестве операций с плавающей точкой в секунду. Но полученный инструмент не подходит для быстрых серийных вычислений в связи со сложностью самой НС.

Berg et al. использовали нейронные сети с глубоким прямым доступом для приближенного решения уравнений в частных производных в сложной геометрии [4]. Они решали уравнения в частных производных адвекционного и диффузионного типа, для которых неприменимы традиционные численные методы. Также провели сравнение между мелкими и глубокими сетями.

В работе Koryagin et al. разработали фреймворк для решения исключительно дифференциальных уравнений в частных производных, обучая нейронную сеть не по известным данным, а в качестве функции, удовлетворяющей уравнению [13]. При этом архитектура нейронной сети должна быть известна на момент поиска решения.

Отсутствие универсальных инструментов для решения дифференциальных уравнений с помощью нейронных сетей вызвано во многом, тем что общая применимость методов машинного обучения к моделированию динамических систем недостаточно изучена. Существует набор работ исследующих результаты использования НС для аппроксимации решений разных типов дифференциальных уравнений.

Также Asady et al. представили применение нейронных сетей для аппроксимации решения линейного двумерного интегрального уравнения Фредгольма второго рода [3]. Они получили высокую точность решения и предложили расширение на случай более общих интегральных уравнений.

В выпускной квалификационной работе студента СПбГУ Виктора Полетанского [21, 9] была исследована применимость методов машинного обучения, в частности нейронных сетей, для решения разных типов дифференциальных уравнений и уравнений в частных производных.

Данная работа является логическим продолжением работы Полетанского.

Исходя из проведённого обзора можно сделать вывод, что проблема быстрой разработки решения дифференциальных уравнений с помощью нейронных сетей существует, а подходящих инструментов для её решения пока что нет, но потенциально возможны. Таким образом актуальным является разработка инструмента, позволяющего предсказывать топологию и параметры обучения нейронной сети по заданному дифференциальному уравнению.

2 Постановка задачи

2.1 Математическая постановка задачи

Представим дифференциальное уравнение в наиболее общей форме:

$$\mathcal{N} f = 0,$$

где $\mathcal{N}$ некий дифференциальный оператор. Решение f дифференциального уравнения вычисляется с помощью нейронной сети:

$$f(x, W) = f(x, W_1, ..., W_n) = \sigma(W_n ... \sigma(W_1 x))$$

Общая архитектура нейронной сети представлена на рисунке 1.

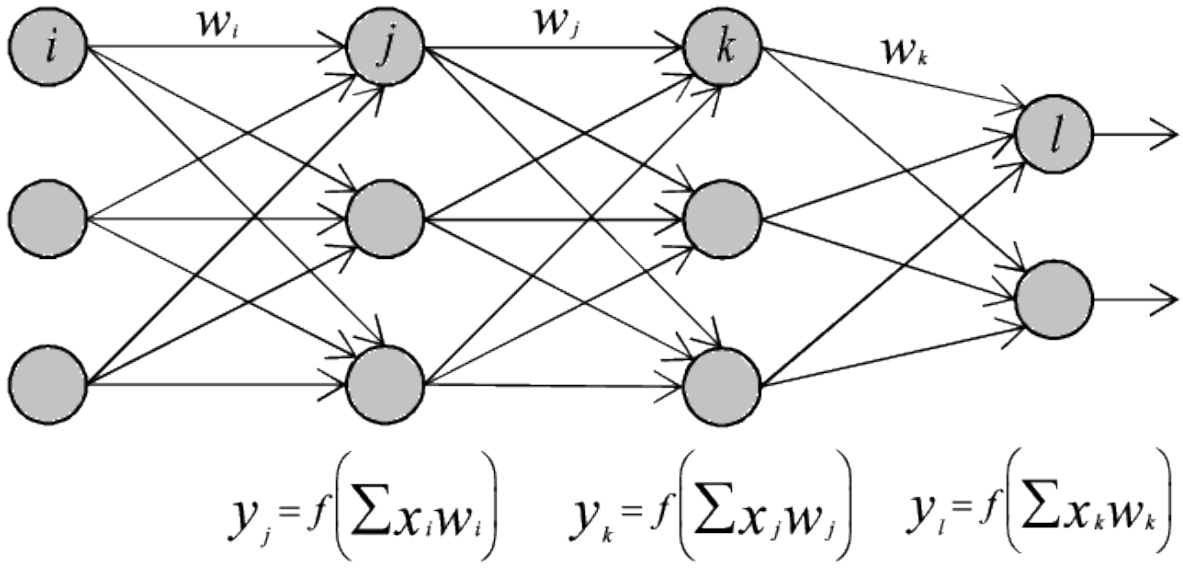

Рис. 1: Представление нейронной сети

Обучение нейронной сети состоит в минимизации выбранной функции потерь. Ниже приведены возможные функции потерь:

$$Loss_{MSE} = \frac{1}{N_f} \sum_{i=1}^{N_f} ||f(t_i) - y_i||^2, \tag{1}$$

$$Loss_2 = \max_{i=1}^{N_f} |f(t_i) - y_i|, \tag{2}$$

где $\{t_1, ..., t_{N_f}\}$ — точки внутри области существования, а $\{y_1, ..., y_{N_f}\}$ — известные значения в соответствующих точках.

Для минимизации функции потерь используется метод градиентного спуска. При вычислении $\mathcal{N}f$ потребуется вычислить производные функции f, то есть производные нейронной сети по входным переменным, что можно сделать с помощью численного дифференцирования.

Для проверки качества обучения нейронной сети используются различные метрики. Улучшение качества обучения состоит в оптимизации метрик. В качестве функции потерь и метрик возможно использование метрик не участвующих в обучении (1)-(2) и следующих функций:

$$Error_1 = \max_{i=1}^{N_f} \begin{cases} 0, |\frac{y_i - f(t_i)}{y_i}| < treeshold \\ |\frac{y_i - f(t_i)}{y_i}|, |\frac{y_i - f(t_i)}{y_i}| \geq treeshold \end{cases} \tag{3}$$

$$Error_2 = \frac{1}{N_f} \sum_{i=1}^{N_f} \begin{cases} 0, |\frac{y_i - f(t_i)}{y_i}| < treeshold \\ |\frac{y_i - f(t_i)}{y_i}|, |\frac{y_i - f(t_i)}{y_i}| \geq treeshold \end{cases} \tag{4}$$

$$Error_{MSLE} = \frac{1}{N_f} \sum_{i=1}^{N_f} ||\log(\mathcal{N}f(t_i))||^2 \tag{5}$$

Где treeshold — порог, отсекающий значения полученные в результате решения, которые достаточно близки к желаемому. Метрика 5 принимает маленькие значения, что позволяет её использовать для работы с жёсткими уравнениями. Метрики 3 и 4 показывают насколько велико максимальное отклонение, отсекая достаточно маленькие колебания, и лежит ли в среднем полученное решение в интервале соответственно.

2.2 Цель

Целью данной работы является создание библиотеки для генерации нейронных сетей по заданным параметрам и оболочки над библиотекой, для заданного дифференциального уравнения предлагает топологию нейронной сети прямого доступа.

Библиотека должна позволять по заданной топологии и параметрам обучения создавать и обучать нейронные сети, осуществлять их

экспорт/импорт в качестве набор весов и смещений, производить генерацию исполняемого кода на C++ для проведения серийных вычислений. Также должна быть возможность проведения экспериментов над обученными нейронными сетями и оценивания их качества и эффективности.

Оболочка над библиотекой должна представлять собой экспертную систему, которая по дифференциальному уравнению, начальному условию и выставленным требованиям к решению предлагает топологию и параметры обучения нейронной сети. Также у оболочки должен быть графический интерфейс.

Для достижения поставленной цели были выделены следующие задачи.

- Задачи на пятый семестр.
  - Выбор инструментария для реализации прототипа библиотеки.
  - Выбор метрик, алгоритмов обучения, функций потерь для обучения нейронных сетей.
  - Разработка прототипа библиотеки.
  - Тестирование прототипа. Создание набора датасетов, выступающих тестами основной библиотеки. Анализ применимости и эффективности архитектуры прототипа в основной библиотеке.
- Задачи на шестой семестр.
  - Разработка архитектуры библиотеки.
  - Создание библиотеки со следующей функциональностью:
    * экспорт нейронных сетей в качестве набора весов и смещений;
    * экспорт нейронных сетей в качестве исполняемого файла на C++;

    * импорт нейронных сетей из набора весов и смещений;
    * генерация нейронных сетей по заданным параметрам;
    * обучение нейронных сетей на заданном наборе данных;
    * реализация оценки качества нейронной сети.
  - Публикация библиотеки в качестве Python пакета.

- Задачи на седьмой и восьмой семестры.
  - Создание датасета для обучения экспертной системы.
  - Разработка экспертной системы, позволяющей:
    * предсказывать топологию нейронной сети по заданному дифференциальному уравнению или системе ДУ;
    * предсказывать параметры обучения.
  - Внедрение экспертной системы в Python пакет библиотеки.
  - Разработка приложения для работы с библиотекой.

3 Обзор методов и инструментов для разработки прототипа

Из существующих языков достаточным количеством функциональности для работы с численными методами и методами машинного обучения обладают не так много языков: Python, C++, Fortran, MATLAB. Так как производительность не сильно важна на этапе обучения нейронных сетей, а MATLAB недоступен на территории Российской Федерации, то Python был выбран, как язык обладающий наибольшим удобством разработки.

В работах Harsha [2], Raissi et al. [16] в качестве инструмента для работы с нейронными сетями используется PyTorch, в работах Rishika et al. [15], Gorikhivskii et al. [9] используется Tensorflow, в работе Rackauckas et al. [20] используется SciML.

BigDL[1] — фреймворк для переноса вычислений на облачные вычислители. Не обладает нужной функциональностью.

Caffe[2] — это среда глубокого обучения, созданная с учетом выразительности, скорости и модульности. Он разработан Berkeley AI Research (BAIR) и участниками сообщества. Не поддерживает технологию GPGPU для видеокарт компании AMD и в настоящее время не разрабатывается.

Chainer[3] — это среда глубокого обучения с открытым исходным кодом, написанная исключительно на Python поверх библиотек Python NumPy и CuPy. Разработку ведет японская венчурная компания Preferred Networks в партнерстве с IBM, Intel, Microsoft и Nvidia. Не поддерживает технологию GPGPU для видеокарт компании AMD и в настоящее время не разрабатывается, также не поддерживает Windows в качестве операционной системы.

Deeplearning4j[4] — это набор инструментов для запуска глубокого обучения на JVM. Варианты использования включают импорт и пере-

[1]Официальный сайт BigDL: https://bigdl.readthedocs.io/en/latest/. Дата посещения: 13.12.2022.
[2]Официальный сайт Caffe: http://caffe.berkeleyvision.org/. Дата посещения: 13.12.2022.
[3]Официальный сайт Chainer: https://chainer.org/. Дата посещения: 13.12.2022.
[4]Официальный сайт Deeplearning4j: https://deeplearning4j.konduit.ai/. Дата посещения: 13.12.2022.

обучение моделей (Pytorch, Tensorflow, Keras), а также развертывание в сервисных средах JVM Micro, мобильных устройствах, IoT и Apache Spark. Не поддерживает технологию GPGPU для видеокарт компании AMD.

Dlib[5] — это набор инструментов C++, содержащий алгоритмы машинного обучения и инструменты для создания сложного программного обеспечения на C++ для решения реальных задач. Не поддерживает технологию GPGPU для видеокарт компании AMD.

Flux[6] — библиотека программного обеспечения для машинного обучения с открытым исходным кодом и экосистема, написанная на Julia. Не поддерживает технологию GPGPU для видеокарт компании AMD.

oneAPI Data Analytics Library[7] — библиотека машинного обучения, которая помогает ускорить анализ больших данных на всех этапах: предварительная обработка, преобразование, анализ, моделирование, проверка и принятие решений. Библиотека реализует классические алгоритмы машинного обучения. Повышение их производительности достигается за счет использования возможностей аппаратного обеспечения Intel. Не поддерживает технологию GPGPU.

Google JAX[8] — платформа машинного обучения для преобразования числовых функций. Описывается как объединение модифицированной версии autograd[9] (автоматическое получение функции градиента путем дифференцирования функции) и TensorFlow XLA[10] (ускоренная линейная алгебра). Поддерживает технологию GPGPU только в рамках видеокарт с технологией CuDa и операционной системы Linux.

Keras[11] — открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с нейронными сетями. В настоящее

[5]Официальный сайт Dlib: http://dlib.net/. Дата посещения: 13.12.2022.

[6]Официальный сайт Flux: https://fluxml.ai/. Дата посещения: 13.12.2022.

[7]Официальный сайт oneAPI Data Analytics Library: software.intel.com/content/www/us/en/develop/tools/data-analytics-acceleration-library.html. На территории РФ недоступен. Официальный репозиторий: https://github.com/oneapi-src/oneDAL. Дата посещения: 13.12.2022.

[8]Официальный сайт Google JAX: https://jax.readthedocs.io/en/latest/. Дата посещения: 13.12.2022.

[9]Официальный репозиторий autograd: https://github.com/HIPS/autograd. Дата посещения: 13.12.2022.

[10]Официальная страница TensorFlow XLA: https://www.tensorflow.org/xla. Дата посещения: 13.12.2022.

[11]Официальный сайт Keras: https://keras.io/. Дата посещения: 13.12.2022.

время является надстройкой над Tensorflow, согласно концепции является интерфейсом. Не самостоятельная библиотека.

MATLAB[12] — пакет прикладных программ для решения задач технических вычислений. Проприетарное программное обеспечение, недоступен на территории Российской Федерации.

Microsoft Cognitive Toolkit[13] — стандартизированный инструментарий для проектирования и развития сетей разнообразных видов, применяет искусственный интеллект для работы с большими объёмами данных путем глубокого обучения, использует внутреннюю память для обработки последовательностей произвольной длины. Не поддерживает технологию GPGPU для видеокарт компании AMD. В настоящее время не разрабатывается.

Apache MXNet[14] — программная среда глубокого обучения с открытым исходным кодом, используемая для обучения и развертывания глубоких нейронных сетей. Библиотека MXNet является переносимой и может масштабироваться на несколько графических процессоров, а также на несколько компьютеров. Не поддерживает технологию GPGPU для видеокарт компании AMD.

Neural Designer[15] — программный инструмент для машинного обучения, основанный на нейронных сетях, основной области исследований искусственного интеллекта, и содержит графический пользовательский интерфейс, упрощающий ввод данных и интерпретацию результатов. Проприетарное программное обеспечение. Не поддерживает технологию GPGPU для видеокарт компании AMD. Поддерживает только аналитическое дифференцирование.

OpenNN[16] — библиотека программного обеспечения, написанная на языке программирования C++, которая реализует нейронные сети, основная область исследований в области глубокого обучения. Не под-

[12]Официальный сайт MATLAB: https://ch.mathworks.com/products/matlab.html. Дата посещения: 13.12.2022.

[13]Официальный сайт Microsoft Cognitive Toolkit: https://learn.microsoft.com/en-us/cognitive-toolkit/. Дата посещения: 13.12.2022.

[14]Официальный сайт Apache MXNet: https://mxnet.apache.org/versions/1.9.1/. Дата посещения: 13.12.2022.

[15]Официальный сайт Neural Designer: https://www.neuraldesigner.com/. Дата посещения: 13.12.2022.

[16]Официальный сайт OpenNN: https://www.opennn.net/. Дата посещения: 13.12.2022.

держивает технологию GPGPU для видеокарт компании AMD.

PlaidML[17] — переносимый тензорный компилятор. Тензорные компиляторы заполняют пробел между универсальными математическими описаниями операций глубокого обучения, таких как свертка, и специфичным для платформы и чипа кодом, необходимым для выполнения этих операций с хорошей производительностью. Не поддерживает технологию GPGPU для видеокарт компаний AMD и NVIDIA.

Apache SINGA[18] — проект Apache по разработке библиотеки машинного обучения с открытым исходным кодом. Он обеспечивает гибкую архитектуру для масштабируемого распределенного обучения, расширяется для работы на широком спектре оборудования и ориентирован на приложения для здравоохранения. Не поддерживает технологию GPGPU для видеокарт компании AMD.

Theano[19] — библиотека Python, позволяющая эффективно определять, оптимизировать и оценивать математические выражения, включающие многомерные массивы. Не поддерживает технологию GPGPU для видеокарт компании AMD. В настоящее время не разрабатывается.

Torch[20] — библиотека машинного обучения с открытым исходным кодом на основе Lua и предоставляет широкий спектр алгоритмов для глубокого обучения. Не поддерживает технологию GPGPU для видеокарт компании AMD. В настоящее время не разрабатывается.

Wolfram Mathematica[21] — программная система со встроенными библиотеками для машинного обучения, статистики, символьных вычислений, сетевого анализа, анализа временных рядов, обработки текстов на естественном языке, функции построения графиков и различные типы данных. Проприетарное программное обеспечение, не поддерживает технологию GPGPU для видеокарт компании AMD.

[17]Официальный репозиторий PlaidML: https://github.com/plaidml/plaidml. Дата посещения: 13.12.2022.

[18]Официальный сайт Apache SINGA: https://singa.apache.org/. Дата посещения: 13.12.2022.

[19]Официальный репозиторий Theano: https://github.com/Theano/Theano. Дата посещения: 13.12.2022.

[20]Официальный сайт PyTorch: http://torch.ch/. Дата посещения: 13.12.2022.

[21]Официальный сайт Wolfram Mathematica: https://www.wolfram.com/mathematica/. Дата посещения: 13.12.2022.

PyTorch[22] — фреймворк машинного обучения для языка Python с открытым исходным кодом, созданный на базе Torch. Используется для решения различных задач: компьютерное зрение, обработка естественного языка. Разрабатывается преимущественно группой искусственного интеллекта Facebook.

Tensorflow[23] — открытая программная библиотека для машинного обучения, разработанная компанией Google. Имеет встроенную поддержку в Google Colaboratory и может работать с TPU модулями.

SciML[24] — аналог библиотеки SciPy для языка Julia. Математическая библиотека с поддержкой автоматического дифференцирования и машинного обучения.

Из описанных инструментов Flux, SciML, Torch, OpenNN не применимы в поставленной задаче, поскольку не имеют интерфейс для Python.

Wolfram Mathematica, Neural Designer, MATLAB являются проприетарным программным обеспечением, а разрабатываемый в данной работе инструмент будет открытым, кроме того использование этих инструментов ограничено на территории Российской Федерации.

BigDL, Keras, oneAPI Data Analytics Library, PlaidML могут быть использованы в качестве вспомогательных инструментов, но не в качестве основного средства работы с нейронными сетями.

Chainer, Caffe, Torch, Theano, Microsoft Cognitive Toolkit в настоящий момент не разрабатываются и не поддерживаются, поэтому также не рассматриваются в качестве подходящего инструмента.

Использование Dlib, Apache SINGA, Apache MXNet, Google JAX, Deeplearning4j возможно, однако их функциональность ограничена, например они не поддерживают технологию GPGPU для видеокарт компании AMD, которая может быть использована для ускорения обучения.

Оставшиеся для рассмотрения библиотеки Tensorflow и PyTorch не

[22]Официальный сайт PyTorch: https://pytorch.org/. Дата посещения: 13.12.2022.

[23]Официальный сайт Tensorflow: https://www.tensorflow.org/. Дата посещения: 13.12.2022.

[24]Официальный сайт SciML: https://sciml.ai/. Дата посещения: 13.12.2022.

обладают описанными выше недостатками, а также использовались в работах других авторов. По предоставляемым возможностям данные библиотеки практически одинаковы, отличия заключаются в том, что PyTorch может работать с OpenMP, в отличии от TensorFlow, но имеет интерфейсы только под три языка, а TensorFlow имеет интерфейсы для девяти языков.

В результате был выбран TensorFlow, так как присутствует необходимость ставить эксперименты и обучать нейронные сети, а данный фреймворк оптимизирован под сервис облачных вычислений Google Colab, и имеет обширную, подробную документацию, большое комьюнити и использовался в работах других авторов, а также в отличии от PyTorch имеет интерфейсы под большое количество языков.

4 Реализация

В данном разделе описывается реализация библиотеки DEGANN, на основе которой в дальнейшем будет реализована экспертная система для предсказывания топологии и параметров обучения нейронных сетей. Также с помощью библиотеки на выбранном множестве дифференциальных уравнений строится набор датасетов с результатами нейронных сетей с целью дальнейшего использования для разработки экспертной системы.

4.1 Прототип

В прошлом семестре был разработан прототип библиотеки, на основе которого определена следующая архитектура библиотеки 2. Во время создания прототипа был выбран Tensorflow в качестве инструмента для работы с нейронными сетями, но благодаря заложенной гибкости архитектуры оказалось достаточно легко использовать другие библиотеки предоставляющие возможности использования НС.

Также в ходе работы над прототипом и последующих экспериментов была получена положительная оценка подходу с использованием нейронных сетей для аппроксимации решений дифференциальных уравнений.

Конкретная функциональность каждого модуля описана ниже.

- Модуль `Trainer Enviroment` отвечает за управление обучением и выбирает из нескольких нейронных сетей ту, которая показала наилучший результат для заданной задачи.

- Модуль `Equation Solver` позволяет решать обычные дифференциальные уравнения первого порядка и системы ОДУ при помощи библиотеки SciPy[25].

- Модуль `Experiment Enviroment` предоставляет окружение для постановки экспериментов, которое по заданным с помощью функ-

[25]Официальный сайт SciPy: https://scipy.org/. Дата посещения: 13.12.2022

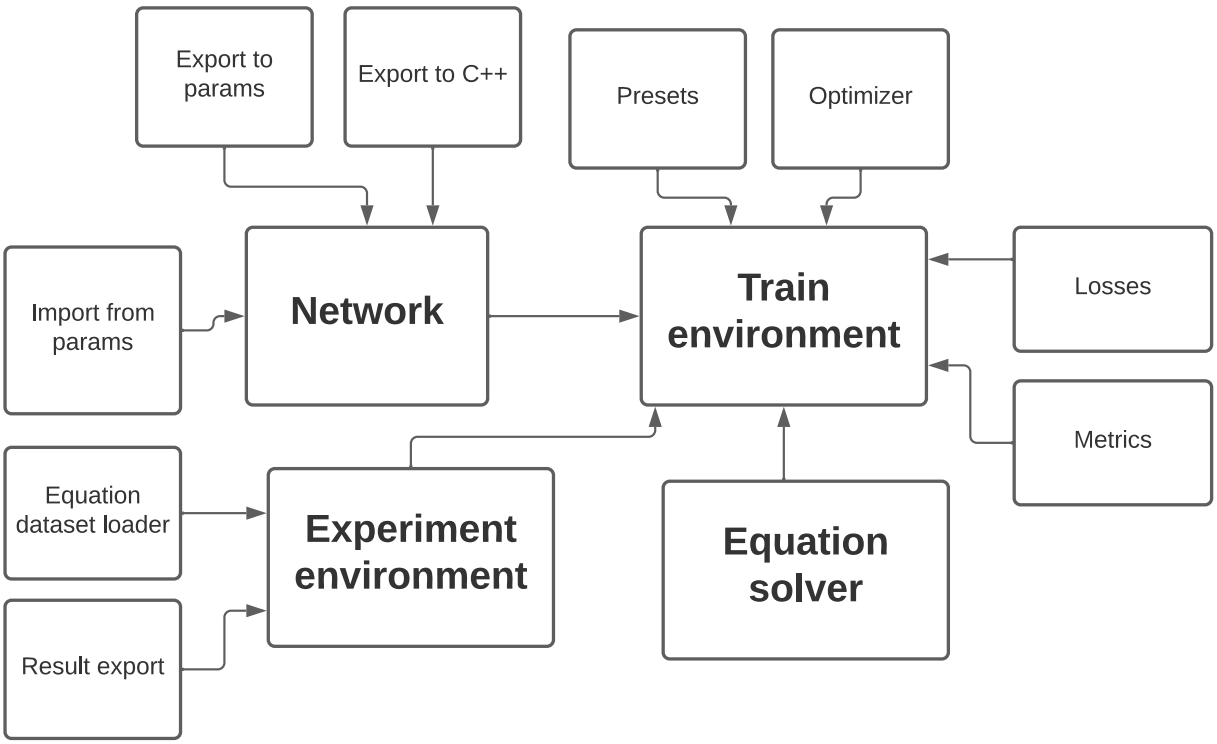

Рис. 2: Общая архитектура библиотеки DEGANN.

ций уравнениям строит таблицы с наборами точек удовлетворяющих решению и экспортирует в файл результаты обучения нейронных сетей.

- Модуль `IModel` является интерфейсом над конкретными реализациями нейронных сетей, в данном случае для `DenseNet`, с которым работают все остальные модули.

4.2 Библиотека DEGANN

В ходе работы над библиотекой архитектура представленная на рисунке 2 была уточнена до диаграммы классов 3.

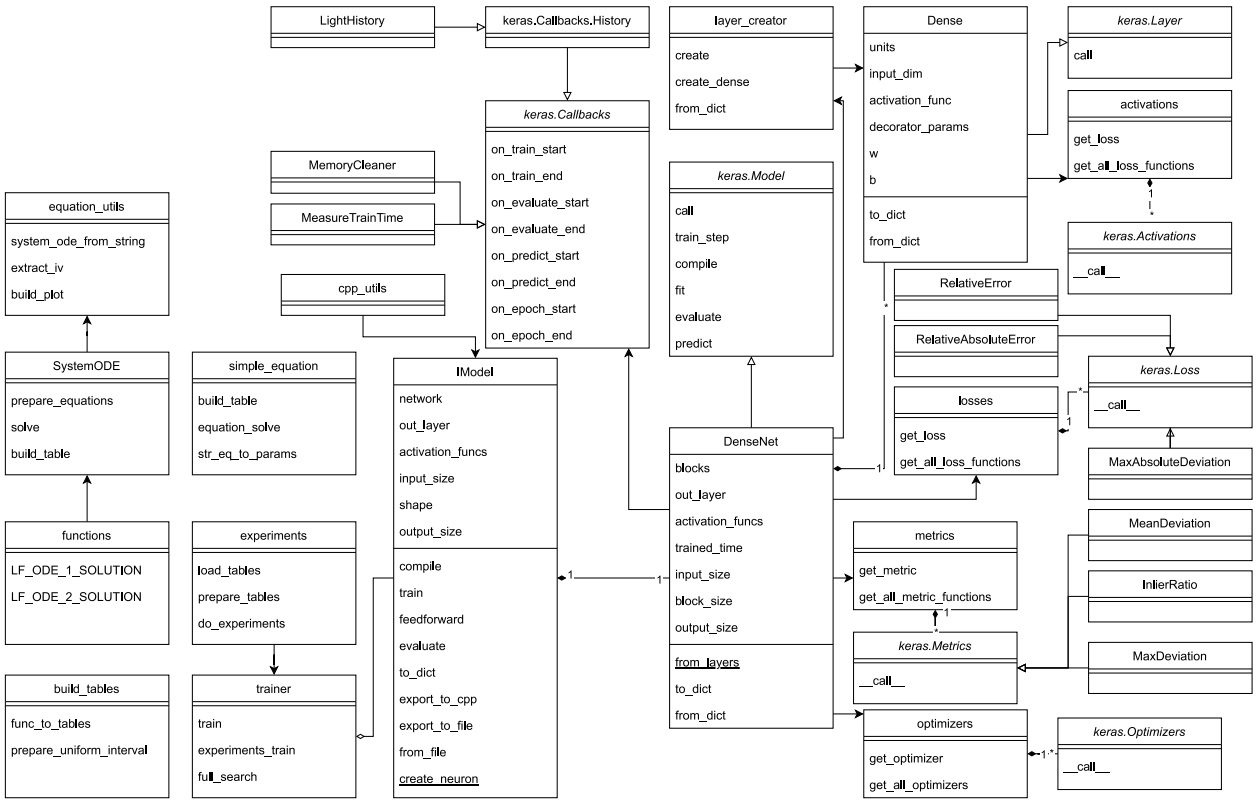

Рис. 3: Архитектура библиотеки.

Модуль `Train Enviroment` представлен на диаграмме классов единственным модулем `trainer`, который позволяет обучать наборы нейронных сетей на входных данных и выбирать лучшую по переданным метрикам. Модуль `Equation Solver` представлен классом `SystemODE`, позволяющем решать обыкновенные дифференциальные уравнения с помощью библиотеки SciPy, и модулем `simple_equation`, который позволяет решать для заданной сетки параметров обыкновенные уравнения с произвольным количеством переменных. Среда для постановки экспериментов `Experimnt Enviroment` представлена модулями `functions`, в котором содержится набор ОДУ, `build_tables`, предоставляющем возможность построения таблиц значений по переданным функциям, и `experiments`, с помощью которого можно собрать данные по обучению нейронных сетей для набор функций и экспортировать результаты в файл. Модуль `IModel` включает в себя класс `IModel`, который служит интерфейсом для работы с нейронными сетями, к примеру с топологией

описанной классом `DenseNet`, основанной на слоях класса `Dense`, также включает функциональность для построения `C++` функций по НС и модули `activations`, `metrics`, `losses` и `optimizers` предоставляющие доступ к различным функциям активации, метрикам, потерям и алгоритмам обучения соответственно.

В настоящий момент можно создавать полносвязные нейронные сети прямого доступа различной топологии и глубины, а также задавать метод обучения, функцию потерь, используемую при вычислении невязки, метрики, активационные функции для слоёв нейронов, шаг обучения, количество эпох и процентный размер валидационных данных.

4.3 Генерация `C++` кода

Библиотека позволяет построить по нейронной сети функцию на `C++`, дающая возможность по вектору входных данных получать ответ НС. Для этого был написан модуль `cpp_utils`, в котором содержится набор параметризуемых функций для определённых создания строк кода на `C++`, к примеру, создание массива, реализация цикла и так далее. Также присутствует возможность создать функцию как в `c-style`, используя ручное управление памятью через `malloc` и указатели, так и в `c++ style`, используя доступный в стандартной библиотеке класс `vector`. После создания кода он экспортируется в качестве `.cpp` и `.hpp` файлов и компилируется с помощью системного компилятора.

Также было проведено сравнение времени работы полученного кода на `C++` и исходного на `Python`. Время измерялось 20 раз для каждой нейронной сети, затем считался доверительный интервал. Замеры проводились на следующем оборудовании.

- Операционная система — `Windows 10 Education`.
- Процессор — `Intel(R) Core(TM) i5-3427U`.
- Среда — `Python 3.10` и `g++ v.9.2.0`.

Результаты представлены в таблице 1, из которой видно, что в результате экспорта нейронных сетей в качестве C++ функции производительность вычислений сильно вырастает.

Таблица 1: Таблица со результатами измерения времени работы метода `feedforward`

Язык	Форма нейронной сети	Размер данных	Время (мс)
Python	10-10-10-10-10-10	500	2539 ± 61
C++	10-10-10-10-10-10	500	2
Python	10-10-10-10-10-10	5 000	24050 ± 628
C++	10-10-10-10-10-10	5 000	35 ± 1
Python	10-10-10-10-10-10	25 000	117882 ± 129
C++	10-10-10-10-10-10	25 000	179 ± 9
Python	100-100-100	500	1659 ± 3
C++	100-100-100	500	56 ± 1
Python	100-100-100	5 000	16621 ± 18
C++	100-100-100	5 000	531 ± 6
Python	100-100-100	25 000	83429 ± 635
C++	100-100-100	25 000	2636 ± 6
Python	500-500-500	500	1828 ± 6
C++	500-500-500	500	1476 ± 30
Python	500-500-500	5 000	18264 ± 63
C++	500-500-500	5 000	14688 ± 144
Python	500-500-500	25 000	91165 ± 81
C++	500-500-500	25 000	73915 ± 1022

Также с помощью сгенерированного кода на C++ можно оценить выигрыш в производительности относительно численных методов. Сравнение проводилось для обыкновенных дифференциальных уравнений описывающих системы химических реакций Робертсона. Для получения данных о времени исполнения численного метода, использовалась библиотека SciPy и метод решения Радау[10], а для данных о времени работы нейронной сети, она сначала обучалась на небольшом количестве известных точек, а затем для получения значений для всего множества точек использовался сгенерированный по НС C++ код. Результаты сравнения представлены в таблице 2 и видно, что относительно численных методов подход с обучением нейронной сети для аппрокси-

мации дифференциальных уравнений, с дальнейшим экспортом в качестве `C++` кода показывает лучшее время.

Таблица 2: Сравнение с численным методом.

Этап	Размер данных	Время (мс)
SciPy, Radau	4 000	64318 ± 727
Python Train	40	13726 ± 105
C++ predict	4 000	39 ± 2

4.4 Тестирование библиотеки

В ходе работы над библиотекой были написаны `unit`-тесты, проверяющие в том числе, что нейронные сети имеют предсказуемый результат, работоспособность экспорт и импорт нейронных сетей и правильность решения дифференциальных уравнений.

При разработке экспертной системы потребуются данные, на которых можно будет проверять её корректность и работоспособность. В этой части работы с помощью функциональности библиотеки был создан набор данных с результатами обучения нейронных сетей, которые были получены при аппроксимации дифференциальных уравнений. Также во время создания датасета было проведено тестирование общей работоспособности библиотеки.

В ходе работы над датасетом были выбраны следующие дифференциальные уравнения.

- Линейные обыкновенные ДУ первого порядка на отрезке $[0, 1]$:

$$y' + 3y = 0,\ y(0) = 1 \qquad \text{(LF-ODE1)}$$

$$y' = 0,\ y(0) = 1 \qquad \text{(LF-ODE2)}$$

$$y' - y = 0,\ y(0) = 1 \qquad \text{(LF-ODE3)}$$

- Линейные обыкновенные ДУ высших порядков на отрезке $[0, 1]$:

$$y'' + 100y = 0,\ y(0) = 0, y'(0) = 10 \qquad \text{(LH-ODE1)}$$

$$y'' + \frac{1}{5}y' + y + \frac{1}{5} * e^{-\frac{1}{5}*x} * \cos(x) = 0,\ y(0) = 0, y(1) = \sin(1)/e^{0.2} \tag{LH-ODE2}$$

- Нелинейные обыкновенные ДУ первого порядка на отрезке $[0.1, 1]$:

$$y' + \frac{y - 2x}{x} = 0,\ y(0.1) = 20.1 \tag{NL-ODE1}$$

$$y' + \frac{y * \cos(x) - 1}{\sin(x)} = 0,\ y(0.1) = 2.1/\sin(0.1) \tag{NL-ODE2}$$

- Системы обыкновенных ДУ первого порядка на отрезках $[0, 1]$ и $[0, \pi]$:

$$\begin{cases} z' + 100y = 0 \\ z - y' = 0 \\ y(0) = 0, z(0) = 10 \end{cases} \tag{SODE1}$$

$$\begin{cases} s' = cd \\ c' = -sd \\ d' = -0.5sc \end{cases} \tag{SODE2}$$

- Жёсткие обыкновенные ДУ:

$$y' + 15y = 0,\ y(0) = 1 \tag{STDE1}$$

$$y'' + 1001'y + 1000y = 0\ y(0) = 1, y'(0) = 0 \tag{STDE2}$$

$$\begin{cases} x' = -0.04x + 10^4yz \\ y' = 0.04x - 10^4yz - 3 * 10^7y^2 \\ z' = 3 * 10^7y^2 \\ x(0) = 1, y(0) = 0, z(0) = 0 \end{cases} \tag{STDE3}$$

Для получения таблиц с данными, удовлетворяющими решению дифференциального уравнения, К уравнениям (SODE2, STDE2, STDE3) применялся численный метод, а для (LF-ODE1)-(SODE1), (STDE1) использовались функции, основанные на известном аналитическом решении.

Были выбраны следующие топологии полносвязных нейронных сетей (в квадратных скобках указаны размеры слоёв).

- $[32, 16, 8, 4]$. Много нейронов на первом слое позволит построить много "низкоуровневых" характеристик, из которых будут строиться несколько "высокоуровневых" характеристик и из них соберется итоговый ответ.

- $[4, 8, 16, 32]$. "Низкоуровневых" характеристик будет меньше, на зато будет больше "высокоуровневых" характеристик для построения решения.

- $[10, 10, 10, 10, 10, 10]$. Сбалансированное количество нейронов в начале и в конце. Также более глубокая связь позволит построить более "сложные" характеристики в конце.

- $[80, 80, 80]$. Широкая неглубокая сеть, может выучить сложные связи, но ее труднее обучать.

Для каждой топологии использовались следующие функции активации (все слои, кроме последнего имеют указанную функцию активации, последний имеет линейную функцию активации).

- *elu*[6] — если аргумент больше нуля, то функция равна линейной, иначе равна $e^x - 1$.

- *relu* — возвращает максимум из аргумента и нуля.

- *gelu* — применяет линейную единицу ошибки Гаусса[11] к аргументу.

- *selu*[19] — если аргумент больше 0, то функция равна линейной с предопределённым коэффициентом 1.05070098, иначе равна $1.05070098 * 1.67326324 * (e^x - 1)$.

- *exponential* — возвращает e возведённую в значение аргумента.

- *linear* — возвращает аргумент неизменным.

- *sigmoid* — по переданному аргументу возвращает значение $\frac{1}{1+e^{-x}}$.
- *hard_sigmoid* — если аргумент меньше 2.5, то значение функции равно нулю, если больше 2.5, то единице, иначе функция возвращает $0.2 * x + 0.5$.
- *swish*[17] — по переданному аргументу возвращает значение $x * \frac{1}{1+e^{-x}}$.
- *tanh* — функция активации гиперболического тангенса — $\frac{(e^x - e^(-x))}{(e^x + e^(-x))}$.
- *softplus* — возвращает значение $\log(e^x + 1)$.
- *softsign* — по переданному аргументу возвращает $\frac{x}{|x|+1}$.

Выбранные нейронные сети учились с различными параметрами обучения на полученных ранее таблицах с решениями. В качестве оптимизаторов использовались Adam[12], SGD и RMSprop. Количество эпох обучения варьировалось от 50 до 200. В качестве функции потерь и метрик использовались среднее квадратичное отклонение, функция потерь хьюбера, максимальное отклонение по модулю, средняя абсолютная ошибка в процентах.

4.5 Результаты экспериментов

В результате тестирования библиотеки появился набор данных, частично представленных в таблице 4. Для каждой конфигурации топологии нейронной сети и параметров обучения в таблицу заносятся значения всех метрик, а также время обучения и предсказывания.

shape	activations	loss_func	optimizer	epochs	loss	train_time
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	Huber	SGD	200	3.564775	10.177364
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	Huber	SGD	200	4.019524	10.181402
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	Huber	Adam	200	3.468842	10.288455
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	Huber	Adam	200	0.003190	10.293120
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	Huber	RMSprop	200	3.469033	10.671893
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	Huber	RMSprop	200	0.546987	10.654902
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanAbsolutePercentageError	SGD	200	56.746208	13.934922
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanAbsolutePercentageError	SGD	200	56.601738	14.641225
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanAbsolutePercentageError	Adam	200	56.540363	16.393556
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanAbsolutePercentageError	Adam	200	3.640137	13.015069
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanAbsolutePercentageError	RMSprop	200	56.789879	25.429137
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanAbsolutePercentageError	RMSprop	200	8.269400	24.360816
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanSquaredError	SGD	200	24.164396	17.479131
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanSquaredError	SGD	200	24.164421	18.342251
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanSquaredError	Adam	200	24.164398	13.820068
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanSquaredError	Adam	200	0.123417	13.529513
[1, 10, 10, 10, 10, 10, 1]	['hard_sigmoid', 'hard_sigmoid', 'hard_sigmoid...	MeanSquaredError	RMSprop	200	24.167412	18.252851
[1, 10, 10, 10, 10, 10, 1]	['swish', 'swish', 'swish', 'swish', 'swish', ...	MeanSquaredError	RMSprop	200	2.522867	18.131187

Рис. 4: Часть набора данных с результатами обучения нейронных сетей.

При этом во время сбора данных был получен вывод, который иллюстрирует рисунок 5, что качество обучения нейронных сетей для аппроксимации решения дифференциальных уравнений сильно зависит от топологии НС и параметров обучения, а также для различных ДУ подходят различные конфигурации. Полученный набор датасетов будет использован для обучения и построения алгоритмов экспертной системы при её разработке.

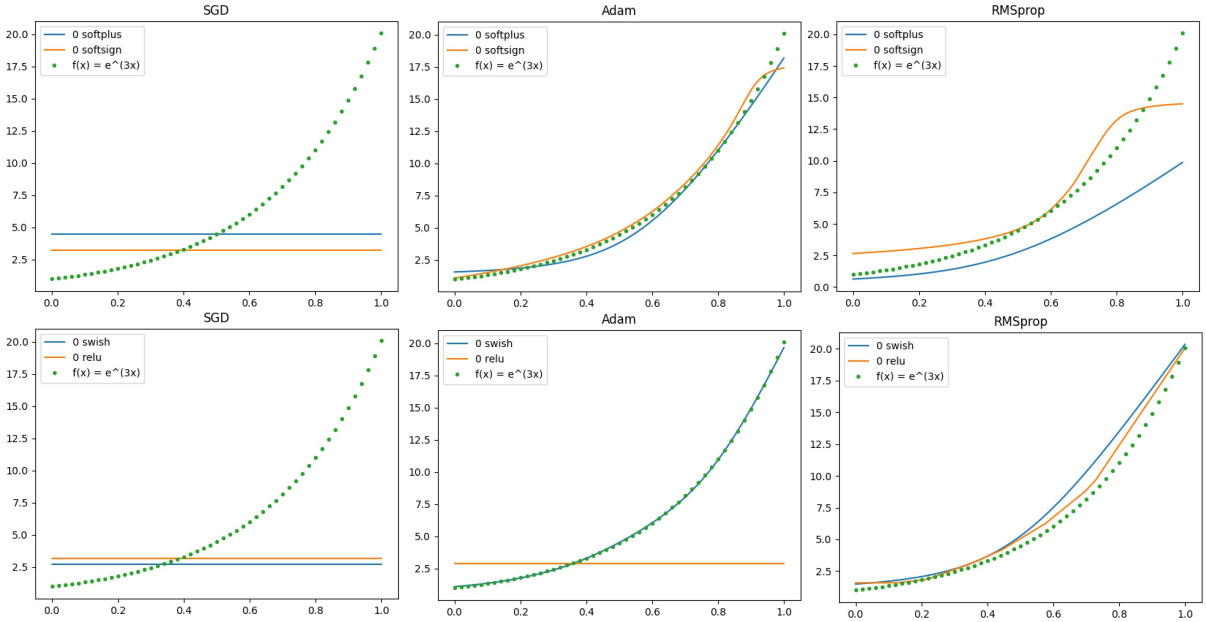

Рис. 5: Качество обучения в зависимости от алгоритма обучения и функции активации.

4.6 Выводы

На основе полученных результатов можно сделать вывод, что библиотека DEGANN успешно справляется с генерацией нейронных сетей и последующим их обучением, по заданным параметрам, на наборе входных данных. Также из результатов, полученных в ходе генерации датасета для экспертной системы, видно, что не для всех комбинаций параметров обучения и топологии нейронной сети действительно находится решение дифференциального уравнения, и это подтверждает необходимость разработки инструмента позволяющего предсказывать конфигурацию НС для аппроксимации решения ДУ.

Заключение

В рамках работы над проектом были реализованы следующие задачи.

- ✓ Разработана архитектура библиотеки DEGANN.
- ✓ Выбраны метрики, алгоритмы обучения, активационные функции и функции потерь для обучения нейронных сетей.
- ✓ Реализована библиотека, позволяющая генерировать нейронные сети для аппроксимации решений дифференциальных уравнений. Также создан Python-пакет, который доступен для скачивания через пакетный менеджер PyPI.
- ✓ На основе библиотеки созданы наборы датасетов, которые будут использоваться при создании алгоритмов экспертной системы.

Библиотека DEGANN и соответствующий Python-пакет для работы с данной библиотекой доступны для скачивания через следующие онлайн ресурсы: https://github.com/Krekep/degann и https://pypi.org/project/degann/

Дальнейшие планы по работе над проектом включают в себя создание экспертной системы и графического интерфейса над библиотекой.

Список литературы

[1] Aksenova Olga A. and Khalidov Iskander A. Simulation of unstable rarefied gas flows in a channel for different Knudsen numbers // AIP Conference Proceedings. — 2019. — Vol. 2132, no. 1. — P. 180009. — https://aip.scitation.org/doi/pdf/10.1063/1.5119667.

[2] Andey Harsha. Deep Neural Networks for solving Differential Equations in Finance. — 2022. — Access mode: https://medium.com/@andeyharsha15/deep-neural-networks-for-solving-differential-equations-in-finar

[3] Asady B., Hakimzadegan F., and Nazarlue R. Utilizing artificial neural network approach for solving two-dimensional integral equations // Mathematical Sciences. — 2014.

[4] Berg Jens and Nyström Kaj. A unified deep artificial neural network approach to partial differential equations in complex geometries // Neurocomputing. — 2018. — Vol. 317. — P. 28–41. — Access mode: https://www.sciencedirect.com/science/article/pii/S092523121830794X.

[5] Bushmakova M. A. and Kustova E. V. Modeling the Vibrational Relaxation Rate Using Machine-Learning Methods // Vestnik St. Petersburg University, Mathematics. — 2022.

[6] Clevert Djork-Arné, Unterthiner Thomas, and Hochreiter Sepp. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). — 2016. — 1511.07289.

[7] Flamant Cedric, Protopapas Pavlos, and Sondak David. Solving Differential Equations Using Neural Network Solution Bundles // CoRR. — 2020. — Vol. abs/2006.14372. — arXiv : 2006.14372.

[8] Gorikhovskii V.I. and Kustova E.V. Neural-Network-Based Approach to the Description of Vibrational Kinetics of Carbon Dioxide // Vestnik St. Petersburg University, Mathematics. — 2022.

[9] Gorikhovskii V I, Evdokimova T O, and Poletansky V A. Neural networks in solving differential equations // Journal of Physics: Conference Series. — 2022. — jul. — Vol. 2308, no. 1. — P. 012008. — Access mode: https://dx.doi.org/10.1088/1742-6596/2308/1/012008.

[10] Hairer Ernst and Wanner Gerhard. Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems. — 1996. — 01. — Vol. 14.

[11] Hendrycks Dan and Gimpel Kevin. Gaussian Error Linear Units (GELUs). — 2020. — 1606.08415.

[12] Kingma Diederik P. and Ba Jimmy. Adam: A Method for Stochastic Optimization. — 2014. — Access mode: https://arxiv.org/abs/1412.6980.

[13] Koryagin Alexander, Khudorozkov Roman, and Tsimfer Sergey. PyDEns: a Python Framework for Solving Differential Equations with Neural Networks. — 2019. — Access mode: https://arxiv.org/abs/1909.11544.

[14] Lei Chon Lok and Mirams Gary R. Neural Network Differential Equations For Ion Channel Modelling // Frontiers in Physiology. — 2021. — Vol. 12. — Access mode: https://www.frontiersin.org/articles/10.3389/fphys.2021.708944.

[15] Mohanta Rishika and Assisi Collins. Parallel scalable simulations of biological neural networks using TensorFlow: A beginner's guide. — 2019. — Access mode: https://arxiv.org/abs/1906.03958.

[16] Raissi M., Perdikaris P., and Karniadakis G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations // Journal of Computational Physics. — 2019. — Vol. 378. — P. 686–707. — Access mode: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[17] Ramachandran Prajit, Zoph Barret, and Le Quoc V. Searching for Activation Functions. — 2017. — 1710.05941.

[18] Mao Guo, Zeng Ruigeng, Peng Jintao, Zuo Ke, Pang Zhengbin, and Liu Jie. Reconstructing gene regulatory networks of biological function using differential equations of multilayer perceptrons // BMC Bioinformatics. — 2022.

[19] Klambauer Günter, Unterthiner Thomas, Mayr Andreas, and Hochreiter Sepp. Self-Normalizing Neural Networks. — 2017. — 1706.02515.

[20] Rackauckas Christopher, Ma Yingbo, Martensen Julius, Warner Collin, Zubov Kirill, Supekar Rohit, Skinner Dominic, Ramadhan Ali, and Edelman Alan. Universal Differential Equations for Scientific Machine Learning. — 2020. — Access mode: https://arxiv.org/abs/2001.04385.

[21] Полетанский В. А. Евдокимова Т. О. Гориховский В. И. Применение методов машинного обучения для решения дифференциальных уравнений. — 2021. — Access mode: http://hdl.handle.net/11701/32448.