МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ
ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский ядерный университет «МИФИ»
Институт Лазерных и Плазменных Технологий
Кафедра № 97 «Суперкомпьютерное моделирование
инженерно-физических процессов»

ОТЧЁТ
Учебная практика (научно-исследовательская работа) на тему:

Построение и исследование нейронной сети с модифицированной функцией потерь и периодической функцией активации для решения дифференциальных уравнений в частных производных

Работу выполнил:
А. Е. Диков
Группа: Б20-221
Научный руководитель:
Д. П. Макаревич

Москва
2024

Содержание

Список используемой литературы **21**

1. Введение

1.1. Актуальность проблемы

В этой работе рассматривается один из современных методов решения дифференциальных уравнений, авторы статьи дали ему оригинальное название - "Physics-Informed Neural Networks" (далее просто PINN), что означает "Нейронные сети, учитывающие физические законы".На сегодняшний день, можно выделить следующие преимущества и особенности данного метода:[4]

- Поскольку дифференциальные уравнения играют ключевую роль в моделировании физических явлений в различных областях, таких как физика, химия, биология, инженерия и финансы, использование PINN позволяет эффективно решать такие уравнения и проводить численные эксперименты в широком диапазоне прикладных задач.

- Во многих случаях сложных физических явлений аналитические решения дифференциальных уравнений недоступны или трудно применимы. Поэтому возникает потребность в разработке эффективных численных методов, одним из которых может быть PINN, способных решать сложные уравнения без явного использования сетки.

- Традиционные численные методы могут требовать большого количества узлов сетки для достижения высокой точности в сложных задачах. PINN предлагает возможность улучшения точности моделирования с меньшим числом узлов или увеличивать число узлов в процессе обучения в областях, где решение имеет большое отклонение по функции потерь, таким образом оптимизируя затраты вычислительных ресурсов.

- В отличие от большинства традиционных методов, PINN является бессеточным методом, где данные о значениях полей не требуются в явном виде на сетке. Это упрощает процесс моделирования и расширяет применимость метода.

- PINN предоставляет возможность интегрировать данные экспериментов в процесс обучения, что делает его мощным инструментом для анализа и моделирования реальных экспериментальных данных в контексте физических задач.

Таким образом, использование PINN для решения дифференциальных уравнений представляет собой перспективное направление исследований, обладающее большим потенциалом для улучшения точности и эффективности численного моделирования в различных областях приложений.

1.2. Трудности реализации задачи

При использовании выбранного мной метода возникают определённые трудности:

- Необходимость самостоятельно выбирать варианты реализации и хранения данных, не существует единственно верного варианта представления задачи

- Метод обладает вариабельностью за счет того, что архитектура в целом является параметром и требует проведения исследования, в ходе которого необходимо сделать выбор число слоев, число нейронов в каждом слое, функцию активации и других параметров.

- Обучение нейронных сетей для решения дифференциальных уравнений может быть сложным из-за нелинейности задачи и высокой размерности пространства параметров. Это может потребовать использования различных методов оптимизации и тщательной настройки гиперпараметров, борьбы с переобучением за счет использования регуляризации, слоев dropout, обучения по пакетам данных - батчам.
- В некоторых случаях PINN может требовать большого объема обучающих данных для достижения высокой точности и обобщения на различные условия, что в совокупности со сложной архитектурой может сделать задачи вычислительно сложной и требующей распараллеливания на графическом процессоре.
- Для более качественной работы PINN требует от разработчика понимания физических процессов, которые моделируются дифференциальными уравнениями. Интеграция физических знаний в процесс построения и обучения сети может потребовать сотрудничества с экспертами в соответствующей области.

1.3. Используемые технологии

Для решения задачи извлечения данных из изображений таблиц используются различные технологии и методы. Ниже приведены некоторые из них:

1.3.1. Глубокое обучение с использованием PyTorch

PyTorch - это библиотека для машинного обучения и глубокого обучения с открытым исходным кодом, позволяющая легко создавать и обучать нейронные сети. Она предоставляет гибкую инфраструктуру для исследований и разработок в области глубокого обучения, а также обширный набор инструментов для работы с тензорами, автоматического дифференцирования и оптимизаций моделей.

1.3.2. Автоматическое дифференцирование

Существует несколько подходов к дифференцированию включая ручные методы, конечные разности, символьные вычисления и автоматическое дифференцирование (АД). Последнее оказывается наиболее удобным для этой задачи, поскольку реализовано в PyTorch и позволяет вычислять градиенты функций и использовать их для оптимизации параметров сети с помощью градиентных методов оптимизации. Основной инструмент для работы с дифференциальными операторами уравнений. Автоматическое дифференцирование включает прямой и обратный проходы. Прямой проход используется для вычисления значений переменных, а обратный проход - для вычисления производных.[2]

Forward pass	Backward pass
$x_1 = 2$ $x_2 = 1$	$\frac{\partial y}{\partial y} = 1$
$v = -2x_1 + 3x_2 + 0.5 = -0.5$ $h = \tanh v \approx -0.462$	$\frac{\partial y}{\partial h} = \frac{\partial (2h-1)}{\partial h} = 2$ $\frac{\partial y}{\partial v} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial v} = \frac{\partial y}{\partial h}\operatorname{sech}^2(v) \approx 1.573$
$y = 2h - 1 = -1.924$	$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_1} = \frac{\partial y}{\partial v} \times (-2) \approx -3.146$ $\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_2} = \frac{\partial y}{\partial v} \times 3 \approx 4.719$

Важно что АД оказывается эффективным при больших размерностях, а применение метода режима Тейлора позволяет вычислять производные высоких порядков с большей точность.[1]

1.3.3. Оптимизационные алгоритмы

Для обучения параметров сети применяются различные оптимизационные алгоритмы, такие как Adam, LBFGS использование которых будет далее описано. Они используют градиенты, вычисленные с помощью автоматического дифференцирования, и обновляют веса модели, минимизируя тем самым функцию потерь.

1.3.4. Ускорение с помощью CUDA

CUDA - это технология, предоставляемая NVIDIA, которая позволяет выполнять параллельные вычисления на графических процессорах (GPU). Графические процессоры обладают множеством ядер (CUDA ядер), которые могут выполнять параллельные задачи. Это дает ускорение работы программ, в которых возможно разбиение на независимые подзадачи. Обучение нейронной сети как раз являются примером такой задачи, поэтому популярные библиотеки для глубокого обучения, такие как PyTorch, предоставляют возможность использования CUDA без необходимости явной реализации. Они автоматически передают операции на GPU при наличии совместимого оборудования.

2. Постановка задачи

Для исследования качества решения и сравнения с другими методами была выбрана задача теплопроводности в двух различных постановках. Первая из них одномерная, наиболее наглядная и позволяет сразу оценить качество решения. На ней тестировались возможные вариации архитектуры нейронной сети.

Постановка первой задачи:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha^2 \dfrac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \quad t > 0, \\ u(x,0) = u_0(x), \\ u(0,t) = u_L(t), \quad u(L,t) = u_R(t) \end{cases}$$

Далее в качестве $u_0(x)$ будет использован $sin(\pi x)$, а температуры на краях будут считаться нулевыми.
Для сравнения с методом конечного элемента была выбрана уже двумерная задача теплопроводности в декартовых координатах

Постановка второй задачи:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha^2 \left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} \right), \quad 0 < x < L_x, \quad 0 < y < L_y, \quad t > 0, \\ u(x,y,0) = u_0(x,y), \\ u(0,y,t) = u_L(y,t), \\ u(L_x,y,t) = u_R(y,t), \\ u(x,0,t) = u_B(x,t), \\ u(x,L_y,t) = u_T(x,t) \end{cases}$$

Постановка задачи для PINN

Каждую из поставленных задач нужно перевести в задачу оптимизации, используя как систему условий. Введем понятие функции потерь.

Определение 1. *В математической статистике и машинном обучении функция потерь $\mathcal{L}$ — отображение результата работы алгоритма на $\mathbb{R}$, показывающее качество работы алгоритма.*

Введем функцию потерь так, чтобы мы могли в отдельности учитывать влияние уравнения, граничных условий и начальных условий.

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}}$$

Теперь подробнее рассмотрим уравнение. Перенесем в нем все слагаемые в одну часть:

$$\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = 0$$

Тогда всякое отклонение уравнения от истины будет нарушать равенство, делать его ненулевым (появляется аналог невязки уравнения):

$$\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = \xi$$

А значит, чтобы уравнение выполнялось, необходимо устремить невязку к нулю, для чего и нужна функция потерь:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left| \frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} \right|^2$$

Здесь был взят определенный вид функции потерь, называемый среднеквадратичной ошибкой (Mean Squared Error, далее просто MSE). Это самый популярный вид, из-за более развитой теории в виде теоремы Гаусса-Маркова и некоторых других утверждений, дающих уверенность в справедливости оценки при выполнении некоторых дополнительных условий.

Теорема 2.1. *Пусть дана линейная модель: $y = X\beta + \varepsilon$, где:*

- *y - вектор наблюдений,*
- *X - матрица регрессоров,*
- *β - вектор параметров модели,*
- *ε - случайный вектор ошибок.*

Тогда, если выполнены следующие предположения:

1. *Линейность: Модель является линейной по параметрам β.*
2. *Ошибки ε являются независимыми и одинаково распределёнными.*
3. *Математическое ожидание ошибок равно нулю, $\mathbb{E}(\varepsilon) = 0$.*
4. *Матрица X полного ранга (столбцы линейно независимы).*
5. *Ошибки имеют постоянную дисперсию (гомоскедастичны).*

Тогда оценки параметров модели, полученные методом наименьших квадратов, являются лучшими линейными несмещёнными оценками с минимальной дисперсией среди всех линейных несмещённых оценок.

Продолжим получение составных слагаемых функции потерь. По аналогии с уравнением переносим все в одну часть и минимизируем эту величину.

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_{\text{IC}}} \sum_{k=1}^{N_{\text{IC}}} |u(x_k, 0) - u_0(x_k)|^2$$

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{\text{BC}}} \sum_{j=1}^{N_{\text{BC}}} |u(x_j, t_j) - u_b(x_j, t_j)|^2$$

В этих формулах $N_{\text{PDE}}, N_{\text{IC}}, N_{\text{BC}}$ - количество точек, соответственно, для условий уравнения теплопроводности, начальных условий и граничных условий. Точки могут быть случайно выбранными в пространстве и времени, использованы все сразу или батчами, меняться в течении обучении или быть постоянными.

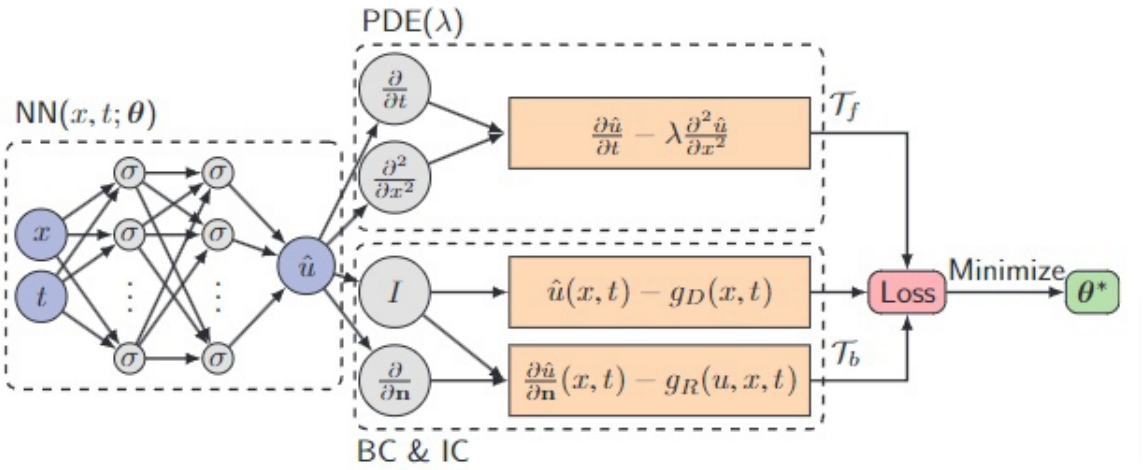

Рисунок 2.1. Схема устройства PINN

3. Анализ предыдущих работ

3.1. DeepXDE: A Deep Learning Library for Solving Differential Equations

В данной статье было подробным образом написано внутреннее устройство PINN, дано общее описание метода, приведены его сильные и слабые стороны. Более важным является то, что результатом работы над этой статьей стала библиотека DeepXDE, предоставляющая набор готовых функций для решения задач теплопереноса с помощью нейронных сетей.

Кроме того, были предложены несколько идей по улучшению метода. Например, добавление весов к потерям точек различных типов, использование оптимизаторов Adam и L-BFGS, проведение настройки параметров под конкретную задачу, увеличение глубины сети и др. Эти идеи были взяты за основу текущей работы.

Авторами было проведено сравнение метода PINN с методом конечного элемента (FEM) по различным характеристикам, но не было представлено численных результатов сравнения.

Метод	**PINN**	**FEM**
Базисные функции	Нейронная сеть (нелинейные)	Кусочно-полиномиальные
Параметры	Веса и смещения	Значения в узлах сетки
Точки	Точки обучения (бессеточный метод)	Точки сетки
Представление ДУ	Функция потерь	Алгебраическая система
Решатель	Оптимизатор	Линейный решатель
Источники ошибок	Аппроксимация, генерализация, оптимизация	Ошибки приближения

Был также описан алгоритм адаптивного уточнения, который уплотняет точки в зависимости от остатков (Residual-Based Adaptive Refinement), что способствует более эффективному обучению.

Algorithm 1 Метод адаптивного уточнения на основе остатков RAR

1: Выберите начальные остаточные точки.
2: Обучите нейронную сеть ограниченное количество итераций.
3: Оцените средний остаток ДУ с использованием метода Монте-Карло. Это делается усреднением значений по набору случайно выбранных уплотненных наборов точек.
4: **if** средний остаток ДУ меньше некоторого порога **then**
5: Завершите процедуру.
6: **else**
7: Добавьте новые точки с наибольшими остаточными значениями из выбранного набора к начальным остаточным точкам.
8: Повторно обучите сеть и вернитесь к Шагу 3.
9: **end if**

3.2. Implicit Neural Representations with Periodic Activation Functions

В статье указано, что современные нейронные сети неспособны моделировать сигналы с высокой степенью детализации и не могут представлять пространственные и временные производные сигнала. Из представленных изображений видно, что предлагаемая авторами смена функции активации дает сильный прирост в качестве.

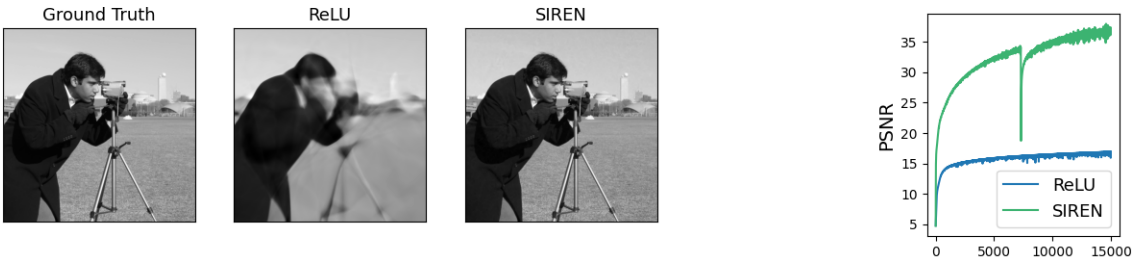

Рисунок 3.1. Пример использования SIREN с изображением

В статье представлены примеры использования этого подхода как для изображений, видео, звука и их производных, так и для решения краевых задач уравнения Пуассона, уравнения Гельмгольца и волновых уравнений.

Авторы статьи предполагают рассмотреть класс функций, где функция $\Phi(\mathbf{x})$ является функцией нейронной сети, задающей взаимно-однозначное соответствие между координатами и, например, цветом на субпиксельном уровне. Полносвязные нейронные сети обеспечивают эффективное по памяти представление объектов. В контексте задачи теплопроводности это было бы соответствие между координатой, временем и температурой.

$$\Phi(\mathbf{x}) = \mathbf{W}_n \left(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0\right)(\mathbf{x}) + \mathbf{b}_n, \quad \mathbf{x}_i \mapsto \phi_i\left(\mathbf{x}_i\right) = \sin\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right)$$

Многослойный перцептрон (Multi layer perceptron, MLP) с гладкими, непериодическими функциями активации не позволяет точно моделировать высокочастотную (то есть детальную) информацию и производные по входным данным.

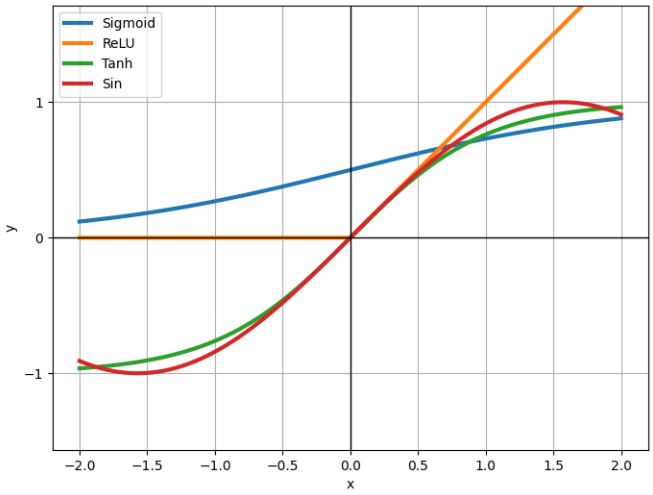

Рисунок 3.2. Наиболее распространенные функции активации

Последний факт связан с тем, что производные от стандартных функций активации либо равны константе или нулю, либо выражаются через первоначальную функцию нелинейно. Например, сигмоида:

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

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

В этом смысле синус имеет преимущество, так как дифференцируем бесконечное число раз и более того, его производная выражается как линейный сдвиг вдоль оси x. Несмотря на то, что уравнение теплопроводности имеет гладкое решение (и сводится к нему за конечное время), периодическая функция активации может дать рост точности за счет своих свойств дифференцируемости. Важным условием корректной сходимости является выбор правильной функции распределения начальных значений весов для линейного слоя, к которому применяется синус. Доказательства приведены в дополнении к статье.[3]

$$\omega_i \sim U(\frac{-c}{\sqrt{n}}, \frac{c}{\sqrt{n}}), \text{ где c} \in \mathbb{R}, \text{ а } n \text{ размерность слоя.}$$

4. Программный код

Опишем основные элементы программы, чтобы обозначить примененные технологии и идеи.

4.1. Входные данные

Зная постановки задачи, было необходимо перевести ее на язык оптимизации. Прежде всего требуется выбрать некоторое облако точек каждого из трех классов (точки уравнения, точки начальных условий, точки граничных условий). Искомые точки могут располагаться регулярно, могут быть выбраны случайно в области решения с различными распределениями вероятности. Получившиеся массивы нужно конвертировать в тензоры и хранить в памяти GPU, чтобы работало CUDA ускорение. Ниже приведен пример функции, для генерации начальных условий, функции для других видов точек выглядят аналогично.

```
1  def set_initial_conditions(initial_distribution_func, size, num_points,
2                             dx, random=False, device="cuda:0"):
3      if random:
4          x_initial = torch.rand(num_points, device=device) * (size -
               2*dx) + dx
5      else:
6          x_initial = torch.linspace(dx, size - dx, num_points,
               device=device)
7
8      t_initial = torch.zeros_like(x_initial, device=device)
9
10     u_initial = initial_distribution_func(x_initial).to(device)
11
12     class InitialConditions:
13         def __init__(self, x, t, u):
14             self.x = x
15             self.t = t
16             self.u = u
17
18     initial_conditions = InitialConditions(x_initial, t_initial,
           u_initial)
19
20     return initial_conditions
```

4.2. Общая структура нейронной сети

Далее объявим основной класс PINN, содержащий все методы, необходимые для обучения, мониторинга качества и применения нейронной сети.

```
class PINN():
    def __init__(self, mesh_params, net_params, initial_conditions,
        boundary_conditions, equation, device='cuda:0'):
        ...

    def network(self):
        ...

    def sample_training_data(self, x, t, u=None, num_samples=None):
        ...

    def function(self, x, t, is_equation=False):
        ...

    def closure(self):
        ...

    def train(self):
        ...

    def predict(self, x, t):
        ...
```

- Метод `__init__` служит для получения данных, на которых будет обучаться сеть, а также для получения параметров архитектуры и создания папки для сохранения результатов.
- Метод `network` непосредственно создает архитектуру нейронной сети, основываясь на полученных параметрах. В нем создается совокупность слоев определенного типа (стандартных линейных или линейных со специальным образом объявленными начальными весами) и задаются функции активации этих слоев.
- Метод `sample_training_data` позволяет обучать модель на различных наборах точек каждую итерацию обучения. Это снижает вычислительные затраты и уменьшает переобучение, но может приводить к сходимости в локальный минимум.
- Метод `function` подает на вход сети точки, при этом получает значения, по которым берет необходимые производные по входным данным, тем самым получая отдельные элементы уравнения.
- Метод `closure` вызывает `function` для разных наборов точек и комбинирует выходные данные так, чтобы получить описанную выше функцию потерь и сделать на основании нее шаг оптимизатора.
- Метод `train` запускает обучение сети в течении обозначенного числа эпох.
- Метод `predict` предоставляет возможность получить предсказание сети в режиме без вычислений градиента.

4.3. Подбор архитектуры

Весь упомянутый выше код, был описан таким образом, чтобы была возможность использовать различные комбинации слоев (их количество и размер), оптимизаторов, функций активации и режимов обучения. Такой подход позволили провести исследование для выявления наиболее подходящей архитектуры для решения поставленной задачи.

```
hidden_layers = np.array([1, 2, 4, 6, 8, 10, 12, 15])
activations = [nn.Tanh(), nn.ReLU(), nn.ELU(), nn.Sigmoid(), "sin"]
neurons_in_layer = np.array([1, 2, 4, 8, 16, 32, 64])
```

5. Текущие результаты

В ходе исследования были проведены эксперименты с различными конфигурациями архитектуры нейронных сетей с целью определения оптимальных параметров. Рассматривались следующие варьируемые параметры:

5.1. Количество скрытых слоев

Исследовались значения от 1 до 15 скрытых слоев.

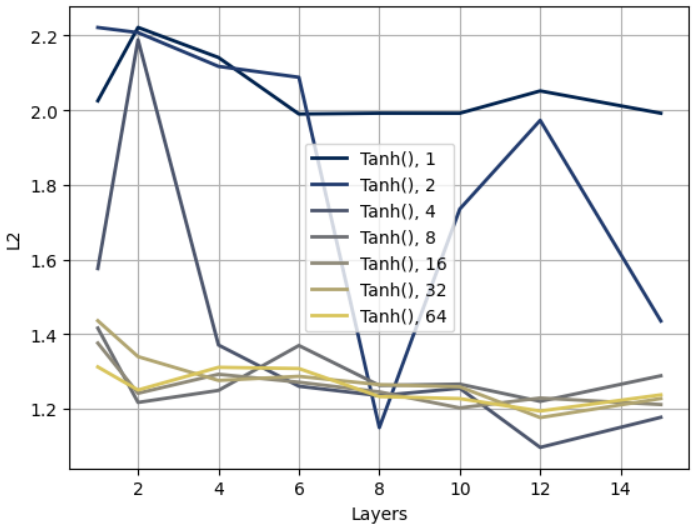

Рисунок 5.1. Зависимость $L2$ меры от числа слоев для функции активации *tanh*

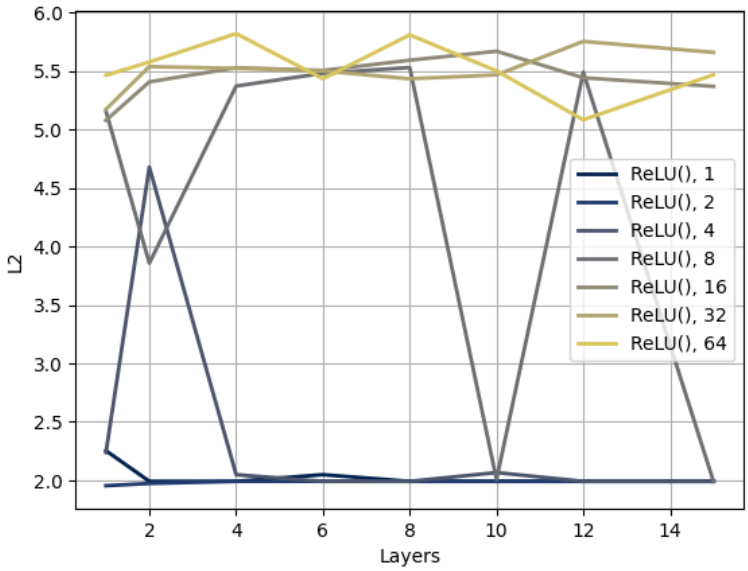

Рисунок 5.2. Зависимость $L2$ меры от числа слоев для функции активации ReLU

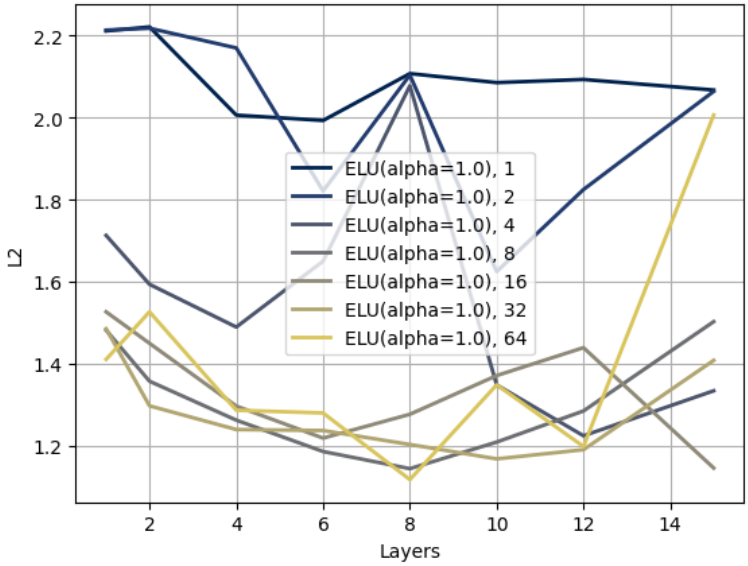

Рисунок 5.3. Зависимость $L2$ меры от числа слоев для функции активации ELU

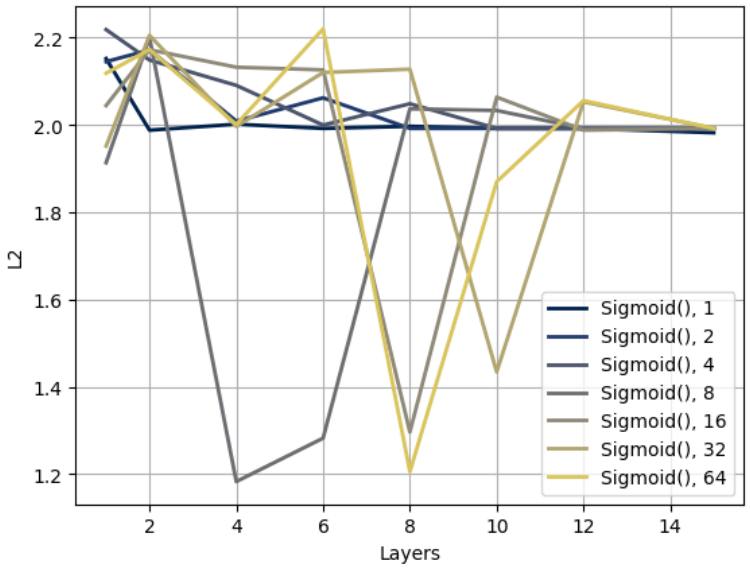

Рисунок 5.4. Зависимость $L2$ меры от числа слоев для функции активации σ

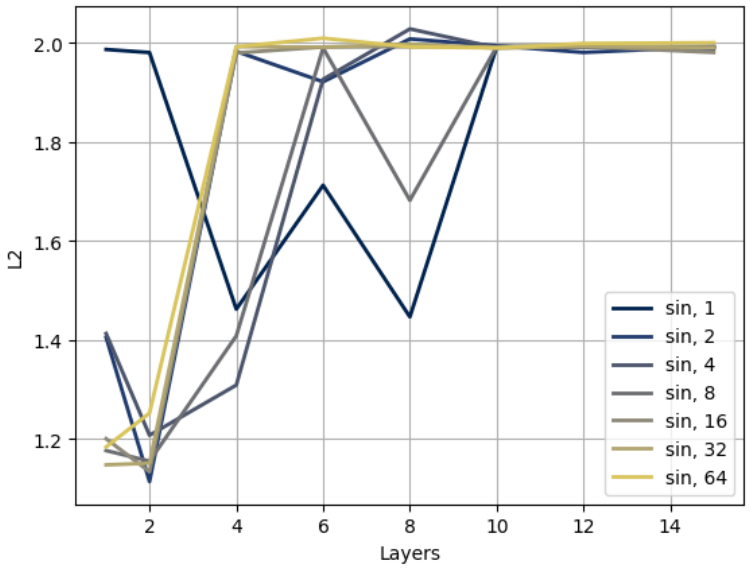

Рисунок 5.5. Зависимость $L2$ меры от числа слоев для функции активации sin

Наблюдается, что увеличение числа скрытых слоев после определенного порога может не приносить существенного улучшения и больше зависит от числа нейронов в слое. Видно, что некоторые функции активации не подходят для поставленной задачи - это происходит из-за того что их вторые производные равны нулю. Важно заметить, что функция активации sin теряет свою точность с ростом числа слоев - этот факт может быть связан с особенностями выбранного оптимизатора.

5.2. Функции активации

Рассмотрены различные функции активации, такие как гиперболический тангенс, ReLU, ELU, σ и sin. Были проанализированы результаты для различных функций активации с целью выявления наилучшей в контексте данной задачи.

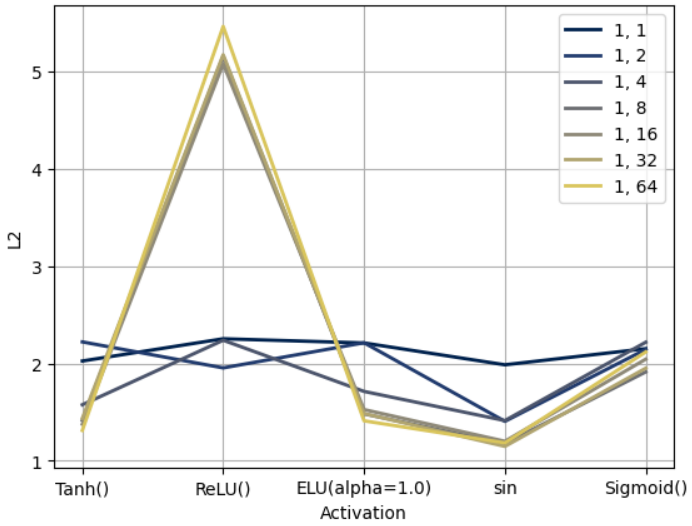

Рисунок 5.6. Зависимость $L2$ меры от функции активации при одном скрытом слое

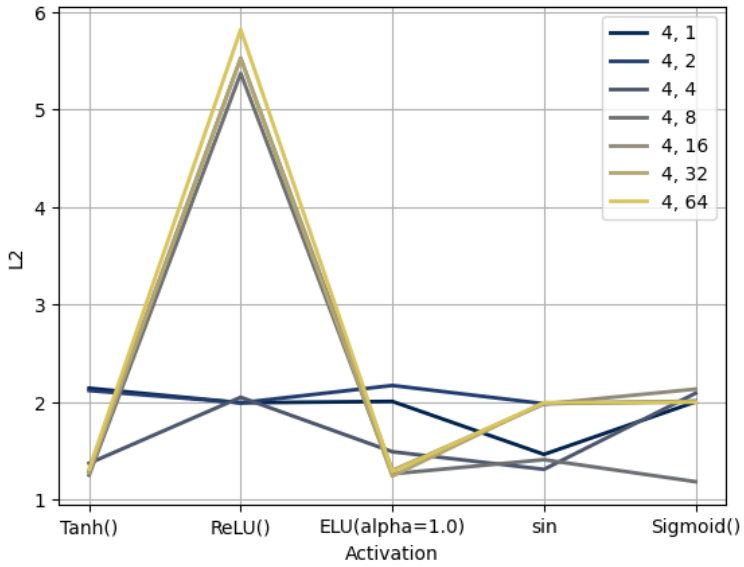

Рисунок 5.7. Зависимость $L2$ меры от функции активации при четырех скрытых слоях

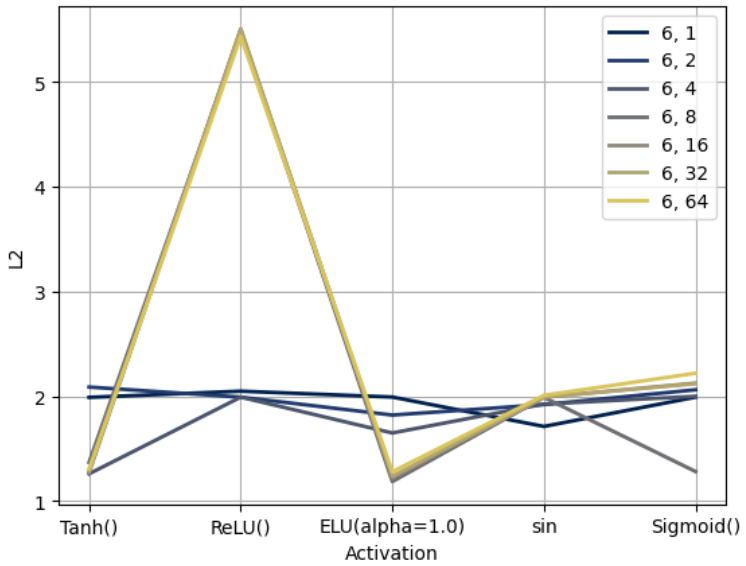

Рисунок 5.8. Зависимость $L2$меры от функции активации при шести скрытых слоях

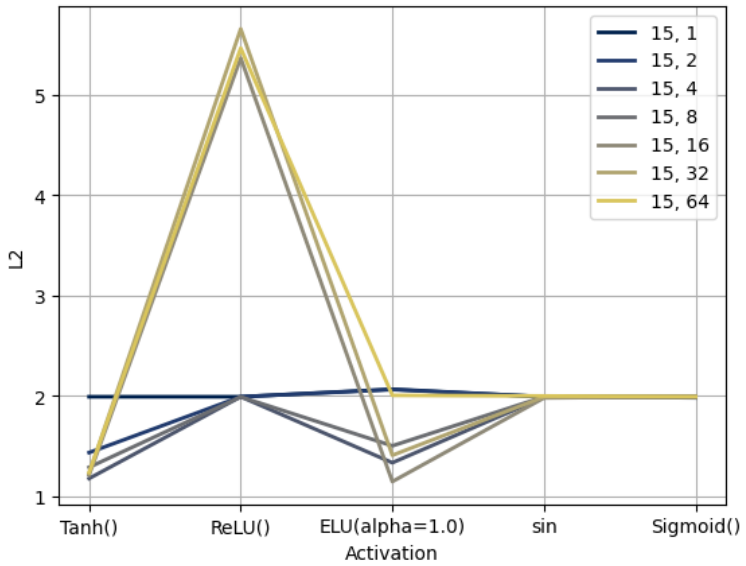

Рисунок 5.9. Зависимость $L2$ меры от функции активации при пятнадцати скрытых слоях

Как видно, *tanh* и ELU показали хорошие результаты для данной задачи. Снова можно заметить что *sin* дает результаты хуже при росте числа слоев.

5.3. Количество нейронов в слое

Проанализировано влияние изменения количества нейронов в скрытых слоях на производительность модели. Рассмотрены значения от 1 до 64 нейронов.

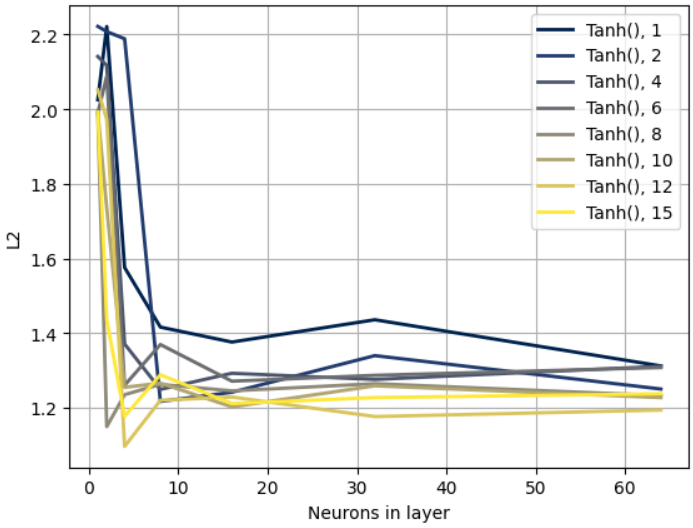

Рисунок 5.10. Зависимость $L2$ меры от числа нейронов в слое для функции активации *tanh*

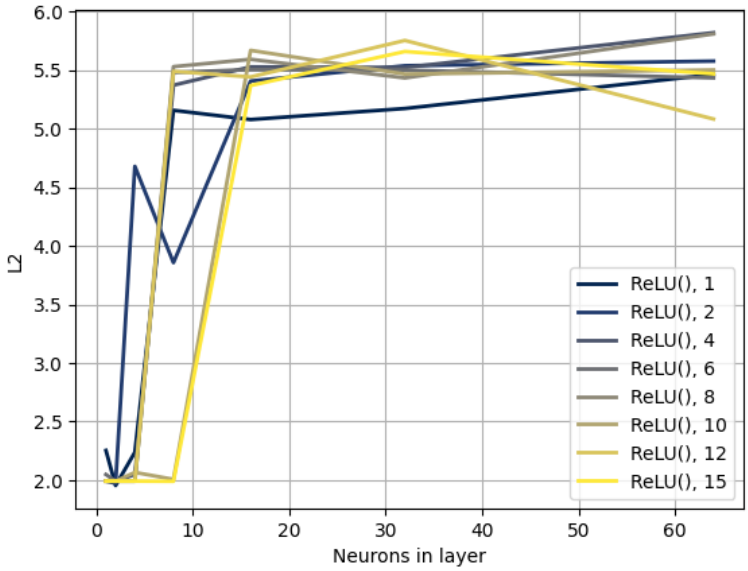

Рисунок 5.11. Зависимость $L2$ меры от числа нейронов в слое для функции активации ReLU

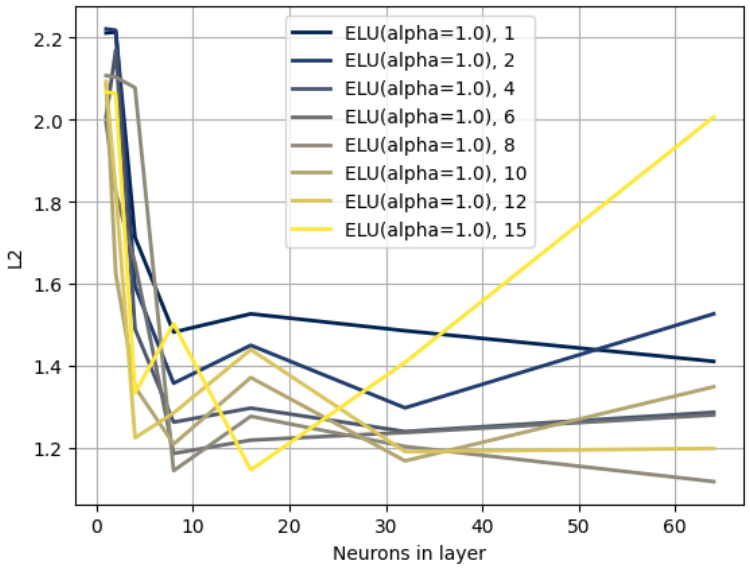

Рисунок 5.12. Зависимость $L2$ меры от числа нейронов в слое для функции активации ELU

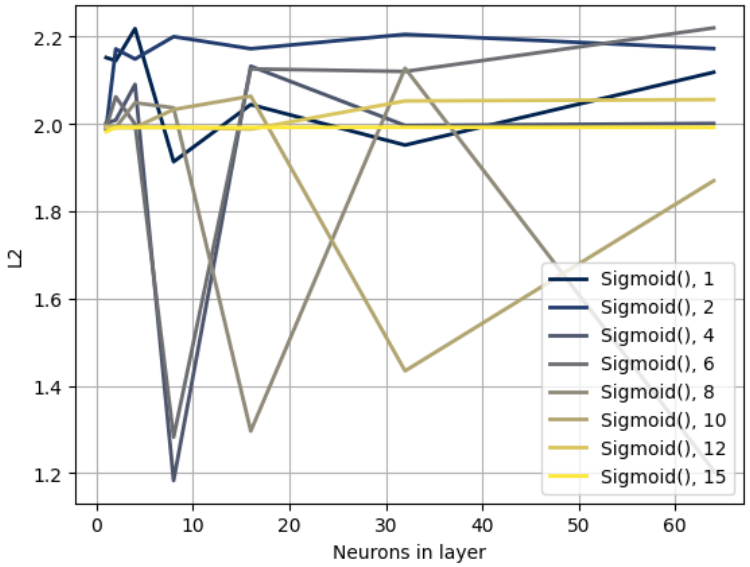

Рисунок 5.13. Зависимость $L2$ меры от числа нейронов в слое для функции активации σ

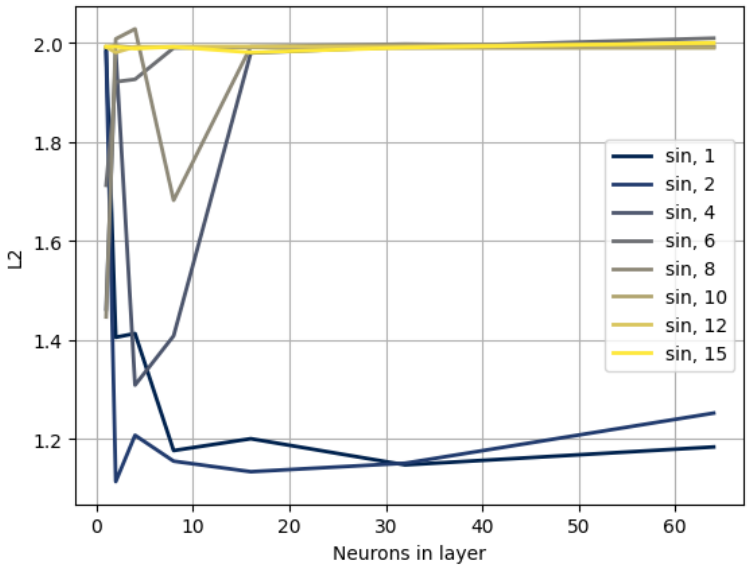

Рисунок 5.14. Зависимость $L2$ меры от числа нейронов в слое для функции активации sin

Как видно из графиков, увеличение количества нейронов в слое даже без увеличения числа слоев приводит к увеличению точности. Однако в случае sin, как уже было отмечено выше, увеличение число слоев ухудшит результат.

6. Планы

Дальнейшие планы по развитию работы строятся следующим образом:

- Необходимо более подробно изучить влияние оптимизатора, сделать сравнение ADAM и L-BFGS или, возможно, использовать их комбинацию. Рассмотреть возможность управления оптимизатором через `scheduler`.
- Изучить влияние сетки и то, где возникает большая ошибка. Сделать ее адаптивной (как было предложено в статье DeepXDE) и на основе этого обновлять веса.
- Запараметризовать функцию потерь, поскольку точки разных типов по-разному влияют на решение. Возможно сделать эту параметризацию адаптивной сделав веса настраиваемыми.

7. Заключение

В ходе исследования и анализа решения дифференциальных уравнений с использованием PINN были получены следующие результаты:

- Произведен обзор современных методов решения дифференциальных уравнений, выявлены их преимущества и недостатки.
- Выявлены трудности и особенности решения дифференциальных уравнений с использованием PINN.
- Определены методы по улучшению качества решений с помощью PINN.
- Рассмотрены перспективы и потенциальные области применения данного метода.

Таким образом, на основе проведенного исследования можно сделать вывод, что метод PINN представляет собой перспективное и эффективное средство для численного решения дифференциальных уравнений в различных областях науки и инженерии.

Список используемой литературы

1. *Bettencourt J.*, *Johnson M. J.*, *Duvenaud D.* Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX // Program Transformations for ML Workshop at NeurIPS 2019. — 2019. — URL: https://openreview.net/forum?id=SkxEF3FNPH.
2. DeepXDE: A Deep Learning Library for Solving Differential Equations / L. Lu [и др.] // SIAM Review. — 2021. — Т. 63, № 1. — С. 208—228. — DOI: 10.1137/19M1274067. — eprint: https://doi.org/10.1137/19M1274067. — URL: https://doi.org/10.1137/19M1274067.
3. Implicit Neural Representations with Periodic Activation Functions / V. Sitzmann [и др.]. — 2020. — arXiv: 2006.09661 [cs.CV].
4. *Raissi M.*, *Perdikaris P.*, *Karniadakis G.* Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations // Journal of Computational Physics. — 2019. — Т. 378. — С. 686—707. — ISSN 0021-9991. — DOI: https://doi.org/10.1016/j.jcp.2018.10.045. — URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.