Применение физически информированного глубокого обучения для решения дифференциальных уравнений

Даниил Лаптев

5 апреля 2024 г.

Содержание

Введение

Physics-Informed Neural Networks (PINNs) [16] обрели популярность в исследовательской среде благодаря, с одной стороны, успехам глубокого обучения для решения самых разных научных и инженерных задач [15, 2], а с другой стороны - огромному количеству вопросов, возникающих в рамках применения этого подхода, и чрезвычайной гибкости его применения. Частный, но достаточно распространённый случай применения этой методологии - решение дифференциальных уравнений и их систем, который мы и рассматриваем в данной работе.

Значительная часть литературы адресует проблему конкретной реализации PINNs, например, для решения тех или иных специфических задач и проблем [1, 8, 7], ускорения их обучения [20] или репрезентации специфических свойств рассматриваемой системы [9, 12, 4]. Кроме того, довольно развита область оценки погрешности и обобщающей способности PINNs [7, 19, 14]. Относительно недавний обзор PINNs и соответствующей литературы можно найти в [3], а также в [5].

Тема гиперпараметров в этой области в большой степени фрагментирована и неоднородна. Этап оптимизации гиперпараметров выступает скорее как инженерная рутина, результаты которой имеют ценность исключительно для какой-либо конкретной ситуации и которая выполняется в полуавтоматическом режиме - больший интерес представляют способы уменьшения числа гиперпараметров, умный дизайн самой PINN и функции потерь. Это не является неожиданностью, поскольку в рамках PINNs существует большое множество различных способов решения той или иной задачи, из-за чего довольно тяжело делать обобщённые выводы.

Тем не менее, в практике обучения PINNs известна серьёзная зависимость качества решения от способа инициализации нейросетевой модели, не в полной мере изучен вопрос выбора оптимальной функции потерь и коэффициентов её термов, не до конца разработан вопрос оптимального выбора точек коллокации для оценки вычисления функции потерь, не вполне ясна роль размера нейросетевой модели, неясно, как ведёт себя нейросетевая модель при обучении и так далее. Литература в этой области существует [13, 18, 6], однако работ, посвящённых общим методологическим выводам и широкому кругу задач, пока не появилось (за исключением некоторых обзорных статей, в которых эти темы практически не обсуждаются). Возникает также ряд теоретических вопросов: например, почему найденные оптимальные гиперпараметры являются оптимальными? Почему для данной задачи необходим такой размер нейросетевой модели и такое число точек коллокации, а для другой задачи - другие? При анализе процесса обучения возникает вопрос о том, как выбрать оптимальный learning rate и каковы особенности ландшафта функции потерь для моделей PINNs [17].

В нашей работе мы предпринимаем попытку начать движение в сторону ответа на эти вопросы, проведя систематическое экспериментальное исследование гиперпараметров на различных задачах и предложив способ классификации сеттинга PINNs. Целью нашей работы является исследование процесса обучения моделей PINNs и влияния гиперпараметров на качество решения. Для достижения цели были поставлены следующие задачи:

1. Реализовать PINNs для решения выбранных дифференциальных уравнений и выявить основные проблемы, которые возникают при их обучении.

2. Выявить связь между гиперпараметрами и качеством решения задачи, проанализировать взаимозависимости между гиперпараметрами.

3. Зафиксировать основные вопросы и проблемы, которые возникают при использовании PINNs, и выдвинуть гипотезы, которые могут разрешить их.

В данной работе мы концентрируемся на решении дифференциальных уравнений (ДУ) с помощью метода PINNs, ограничившись классическим вариантом непрерывного PINNs, который описан в работе [16]. Кроме того, наша реализация методологии использует так называемые мягкие ограничения (soft constraints), которые, в отличие от жёстких ограничений (hard constraints), позволяют нейросетевой модели совершать ошибки на границе домена [3]. Тема обнаружения ДУ, а также различных расширений метода остаётся вне поля нашего внимания.

Обзор литературы

1 Методология Physics-Informed Neural Networks

Рассмотрим общую постановку проблемы. Решением ДУ называется функция u, определённая на пространстве Ω, которая удовлетворяет условиям:

$$\begin{aligned} \mathcal{D}u &= f, \\ \mathcal{B}u &= g, \end{aligned} \tag{1}$$

где $\mathcal{D}$ - дифференциальный оператор для всего домена, $\mathcal{B}$ - дифференциальный оператор на границе домена, f и g - какие-либо заданные функции.

Опираясь на свойство универсальной аппроксимации [10], мы можем пожелать приблизить искомую функцию некоторой нейросетевой моделью с параметрами θ. Тогда проблема решения ДУ сведётся к задаче оптимизации (обучения) относительно какой-либо функции потерь.

Пусть $\mathcal{N}(x;\theta)$, где $x \in \Omega$ - нейросетевая модель. Для удобства будем писать $\mathcal{N}_\theta$. То, насколько нейросетевая модель удовлетворяет условиям (1), можно выразить следующим образом:

$$\begin{aligned} L_\mathcal{D}(\theta) &= \|\mathcal{D}\mathcal{N}_\theta - f\|, \\ L_\mathcal{B}(\theta) &= \|\mathcal{B}\mathcal{N}_\theta - g\|, \end{aligned} \tag{2}$$

Тогда оптимизируемую функцию потерь можно записать как

$$\mathcal{L}(\theta) = w_1 L_\mathcal{B}(\theta) + w_2 L_\mathcal{D}(\theta), \tag{3}$$

где коэффициенты w_1 и w_2 нужны для того, чтобы балансировать между важностью граничного и внутреннего термов функции потерь. В тех или иных условиях может потребоваться пожертвовать точностью на границе для того, чтобы улучшить точность внутри домена, и наоборот. Для удобства мы введём параметр C и определим $w_1 = C$, $w_2 = (1-C)$, как в работе [13], тем самым уменьшив число гиперпараметров на один.

В конечном итоге для какой-либо фиксированной нейросетевой архитектуры мы ищем такой набор параметров θ^*, что

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$$

Рассмотрим (2) и (3). Минимизацию функции потерь $L_\mathcal{B}$ можно воспринимать как классическое обучение с учителем, поскольку на границе нам известны значения искомой функции и/или её производных; в свою очередь, $L_\mathcal{D}$ является регуляризационным термом, который ограничивает пространство допустимых параметров. L1 и L2 регуляризация направляет оптимизатор в те области, в которых норма параметров будет небольшой; регуляризация с помощью $L_\mathcal{D}$, который часто называется PINN-термом функции потерь, направляет оптимизатор в те области, в которых нейросетевая модель будет удовлетворять дифференциальному уравнению внутри домена. Иными словами, мы ищем параметры нейросети в таком множестве, которое можно описать следующим образом:

$$\Theta = \{\theta : \quad \|\mathcal{D}\mathcal{N}_\theta - f\| \approx 0 \quad \text{и} \quad \|\mathcal{B}\mathcal{N}_\theta - g\| \approx 0\}$$

Поскольку на практике мы не можем достичь равенства нулю, мы используем знак $\approx$. В разных контекстах близость к нулю может пониматься по-разному. Если решение поставленной задачи существует и единственно, то минимизация описанных выше функций потерь означает также и приближение искомой функции, причём ровно одним способом (подробнее см. в разделе 4).

Чтобы получить значение функции потерь, обычно используется Монте-Карло оценка L2 нормы, по-другому известная как Mean Squared Error. Например, если на границе нами выбрано (или нам дано) $N_{\mathcal{B}}$ пар вида (x_i, g_i), а внутри домена мы каким-либо образом выбираем $N_{\mathcal{D}}$ точек x_i, то функцию ошибки можно оценить как

$$L_{\mathcal{D}}(\theta) = \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} (\mathcal{D}\mathcal{N}_\theta(x_i) - f(x_i))^2,$$
$$L_{\mathcal{B}}(\theta) = \frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} (\mathcal{B}\mathcal{N}_\theta(x_i) - g(x_i))^2$$

Формально разницы между этими оценками нет, однако нередко на границе заданы сами значения искомой функции (например, если граничные условия поставлены в форме Дирихле) и/или её производных (например для задачи Коши, в которой известно значение функции и её производных в начальный момент времени). Интуитивно мы можем понимать $L_{\mathcal{B}}$ как условие на точное значение искомой функции, а $L_{\mathcal{D}}$ как условие на характер поведения искомой функции на всём домене.

Дифференциальный оператор, как граничный, так и внутренний, легче всего приблизить с помощью алгоритма автоматического дифференцирования. Тогда частные производные любого порядка могут быть вычислены для любой точки, в которой мы оцениваем значение нейросетевой модели. Единственное условие - наличие функции активации подходящей гладкости, чтобы производные высших порядков вообще существовали.

Процесс оптимизации чаще всего реализуется с помощью алгоритмов Adam и L-BFGS [3], нередко в комбинации - сначала происходит тренировка с помощью Adam, затем тренировка продолжается с помощью L-BFGS. В нашем исследовании мы ограничимся алгоритмом Adam.

В качестве нейросетевой архитектуры мы используем модели прямого распространения (FF), которые представляются как композиция аффинных и нелинейных преобразований:

$$\begin{aligned} Linear_i(x) &= W_i x + b_i, \\ FF(z) &= Linear_L(...\phi(Linear_1(z))), \end{aligned} \tag{4}$$

где ϕ - некая функция активации.

Для наших экспериментов мы возьмём такие модели, у которых количество строк в матрицах $W_1, ..., W_{L-1}$ одинаково. Пусть это количество (ширина нейросети) записывается как W. Тогда модель глубины L и ширины W будет обозначаться как $\mathrm{FF}_{L,W}$. Если каждый параметр в нейросети пронумерован и всего их p, то вектор $\theta \in \mathbb{R}^p$ можно взаимно-однозначно соотнести с параметрами нейросетевой модели.

В общем случае нейросетевые модели могут быть произвольными.

2 Решение дифференциальных уравнений

Свойство универсальной аппроксимации лишь гарантирует нам потенциальную возможность аппроксимировать искомую функцию некоторой нейросетевой моделью, однако для того, чтобы это сделать, необходимо подобрать подходящий алгоритм обучения. Последний мы понимаем как совокупность нейросетевой модели, формулировки функции потерь,

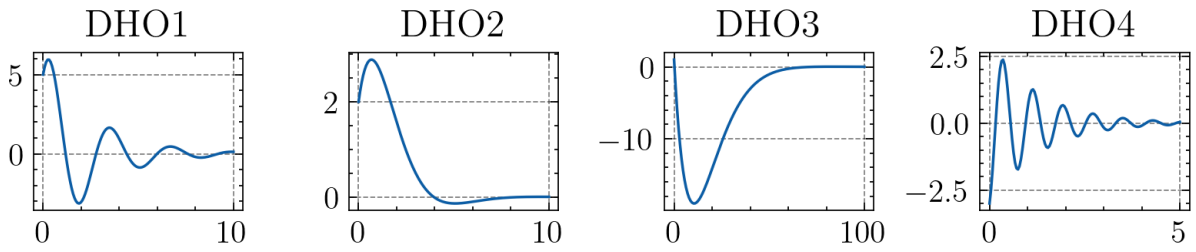

Рис. 1: Различные варианты поведения решений гармонического осциллятора.

Параметры	DHO1	DHO2	DHO3	DHO4
ζ, ω_0	0.2, 2.0	0.7, 1.0	0.9, 0.1	0.1, 8
x_0, v_0	5.0, 7.0	2.0, 3.0	1.0, -5.0	-3.0, 10.0
T	10	10	100	5

Таблица 1: Параметры постановки задач для системы (5).

а также гиперпараметров тренировочного процесса. В данном разделе мы опишем простейший способ решения некоторых ДУ с помощью PINNs, и сфокусируемся на проблемах, которые возникают в процессе обучения и выбора оптимальных гиперпараметров.

Для того, чтобы оценивать качество PINNs, мы использовали и реализовали некоторые классические численные методы для получения ground truth значений искомой функции (подробнее см. в разделе A). Чем точнее будет получено решение ДУ, тем точнее будет оценка качества работы нейросетевой модели.

2.1 Гармонический осциллятор с затуханием

Система описывается следующим образом:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} + 2\zeta\omega_0\frac{\mathrm{d}x}{\mathrm{d}t} + \omega_0^2 x = 0, \tag{5}$$

где функция $x(t) : \mathbb{R} \to \mathbb{R}$ описывает положение осциллирующей массы, а параметры ζ и ω_0 влияют на характер осцилляции и её затухания.

Пусть дано начальное положение массы $x(0) = x_0$ и начальная скорость $v(0) = \frac{\mathrm{d}x}{\mathrm{d}t}(0) = v_0$. Зафиксируем некоторую нейросетевую архитектуру $\mathcal{N}(z;\theta)$. Тогда мы должны отыскать такие параметры θ, которые минимизируют следующий функционал:

$$L(\theta) = \frac{C}{2}\left\| \begin{bmatrix} \mathcal{N}_\theta(0) - x_0 \\ \frac{\mathrm{d}\mathcal{N}_\theta}{\mathrm{d}t}(0) - v_0 \end{bmatrix} \right\|_2^2 + \frac{1-C}{N_\mathcal{D}} \sum_{i=1}^{N_\mathcal{D}} \left(\left[\frac{\mathrm{d}^2}{\mathrm{d}t^2} + 2\zeta\omega_0\frac{\mathrm{d}}{\mathrm{d}t} + \omega_0^2 \right] \mathcal{N}_\theta(t_i) \right)^2$$

Графики рассматриваемых решений приведены на рисунке 1, а параметры постановки задач приведены в таблице 1.

Ручной подбор гиперпараметров оказался достаточно трудоёмким, из-за чего было принято использовать автоматический подбор гиперпараметров (подробнее см. в разделе B). Далее мы называем его алгоритмом HPO, который с течением времени постепенно сходится в область оптимальных гиперпараметров. В результате проведённых экспериментов оказалось, что при увеличении времени T необходимо увеличивать размер нейросетевой модели. Неясно, однако, каково влияние остальных параметров постановки задачи на оптимальные гиперпараметры, поскольку, например, задача DHO3 потребовала довольно мало точек коллокации, несмотря на то, что домен решения значительно увеличился.

Точки коллокации мы сэмплируем один раз перед началом обучения. Производим мы это двумя способами - либо выбираем точки равноудалёнными друг от друга на отрезке $[0, T]$, формируя сетку, либо берём значения случайной величины из равномерного распределения на $[0, T]$. Для каждой задачи среди лучших результатов чаще всего встречается первый метод. Это значит, что алгоритм HPO счёл разбиение по сетке наиболее оптимальным.

Нейросетевые модели смогли практически идеально повторить форму решения задач DHO1, DHO2 и DHO3, однако в задаче DHO4 динамика искомой функции была слабо выучена в правой половине домена (рисунок 2). Мы предполагаем, что это связано в первую очередь с тремя факторами: а) недостаточно долгое обучение, б) затухающий градиент нейросетевой модели при $t \to \infty$, и в) относительно слабая динамика и соответственно близкая к нулю производная в правой части домена.

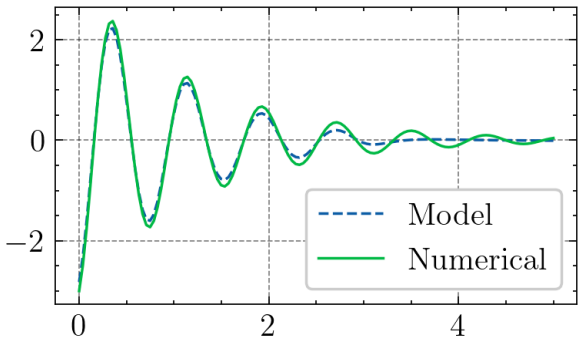

Рис. 2: Решение задачи DHO4.

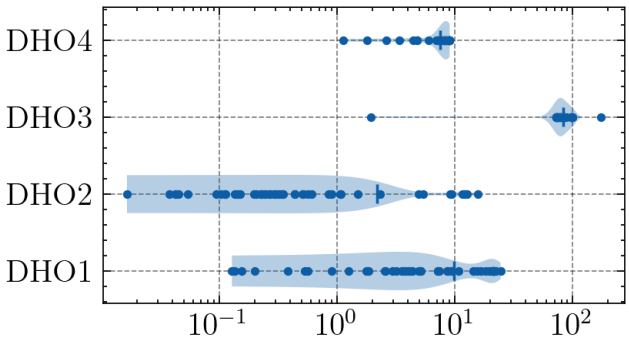

Рис. 3: Распределение значений L2 ошибки.

Несмотря на общую простоту решения этой системы, мы нередко наталкиваемся на различные проблемы, связанные с процессом обучения. Например, задача DHO3 решилась, вообще говоря, случайным образом. На рисунке 3 представлено распределение значения ошибок. Для каждой задачи было взято по 50 случайно выбранных гиперпараметров. Видно, что в задачах DHO3 и DHO4 мы редко опускались до низких значений ошибки, и, более того, для DHO3 второе наименьшее значение равно 72.5. Можно сделать вывод, что в данном случае оптимизация гиперпараметров представляет собой довольно нетривиальную задачу, с которой даже автоматический алгоритм может не справиться с первого раза за разумное время.

Возникает следующий вопрос: почему так выходит, что увеличение длины домена $[0, T]$ приводит к усложнению обучения, даже несмотря на то, что сама искомая функция имеет довольно простую форму? Наилучшим объяснением этому является то, как нейросетевая модель рассчитывает значения функции в тех или иных точках, а также то, как считаются градиенты для расчёта PINN-терма и градиентов весов. В первую очередь проблемы могут возникать из-за того, что функция активации tanh, используемая нами в архитектуре модели, асимптотически ограничена сверху и снизу, и её производная при больших положительных и отрицательных значениях аргумента практически равна нулю. Для того, чтобы работать с большими величинами (большим t и большими значениями функции), нейросеть должна настроить веса входного и выходного слоя таким образом, чтобы уметь обрабатывать их так же, как она обрабатывает малые величины. Кроме того, нулевое значение производной функции активации практически на всём домене может привести к тому, что нейросеть просто не будет иметь информацию о том, как ей следует правильно обучаться, что приведёт к торможению обучения и плохой аппроксимации. Соответственно, удачная аппроксимация, которую мы смогли достичь - это скорее всего результат удачной инициализации весов, которая помогла нейросети обучиться и не застрять с затухающими градиентами. Этот вопрос требует дальнейшего анализа.

Кроме того, мы выявили, что режимы работы PINN можно разбить на два класса - хорошая аппроксимация, при которой нейросеть постепенно сходится к искомой функции и вопросом является лишь то, как ускорить сходимость, и плохая аппроксимация, при которой нейросетевая модель начинает приближать нулевую функцию (или константную) либо на всём домене целиком, либо на большей его части. Такое поведение возникает в разных ситуациях, и неясно, что именно является причиной и как заранее не допустить попадания нейросети в режим плохой аппроксимации.

Мы также выявили, что обучение происходит, так сказать, слева направо. Если визуализировать процесс обучения [1], то мы обнаружим, что нейросеть аппроксимирует искомую функцию постепенно, сначала повторяя динамику в левой части домена, затем в правой. Особенно это заметно для DHO1 и DHO4. Такое поведение наблюдается и для других ДУ.

2.2 Система Лотки-Вольтерры

Рассмотрим модель взаимодействия двух биологических видов, один из которых выполняет роль хищника, другой - жертвы. Такая система часто моделируется с помощью системы ОДУ Лотки-Вольтерры. Пусть число особей в их популяциях описывается соответственно непрерывными функциями $y(t)$ и $x(t)$. Тогда динамика числа особей будет выражена как

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy \\ \frac{\mathrm{d}y}{\mathrm{d}t} = \delta yx - \gamma y \end{cases}, \tag{6}$$

где параметры $\alpha, \beta, \delta, \gamma$ характеризуют рождаемость и смертность популяций.

Пусть нейросетевая модель $\mathcal{N} : \mathbb{R} \to \mathbb{R}^2$ будет аппроксимировать значение сразу двух искомых функций в момент времени t посредством двумерного вектора на её выходе:

$$\mathcal{N}(t;\theta) = \begin{bmatrix} \mathcal{X}(t;\theta) \\ \mathcal{Y}(t;\theta) \end{bmatrix} \approx \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$$

Пусть даны начальные значения x_0, y_0. Нам необходимо найти такие параметры θ, которые минимизируют следующие функции ошибки:

$$\begin{aligned} \mathcal{L}_{\mathcal{I}} &= \frac{1}{2} \left\| \begin{bmatrix} \mathcal{X}(0) - x(0) \\ \mathcal{Y}(0) - y(0) \end{bmatrix} \right\|_2^2 \\ \mathcal{L}_{\mathcal{X}} &= \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \left(\frac{\mathrm{d}\mathcal{X}}{\mathrm{d}t}(t_i) - \alpha \mathcal{X}(t_i) + \beta \mathcal{X}(t_i)\mathcal{Y}(t_i) \right)^2 \\ \mathcal{L}_{\mathcal{Y}} &= \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \left(\frac{\mathrm{d}\mathcal{Y}}{\mathrm{d}t}(t_i) - \delta \mathcal{X}(t_i)\mathcal{Y}(t_i) + \gamma \mathcal{Y}(t_i) \right)^2 \end{aligned} \tag{7}$$

В качестве итоговой функции потерь мы возьмём их линейную комбинацию:

$$L_{\mathcal{T}} = C L_{\mathcal{I}} + (1 - C)(L_{\mathcal{X}} + L_{\mathcal{Y}})$$

На рисунке 4 представлены рассматриваемые нами задачи, а в таблице 2 приведены параметры постановки задач. Мы вновь выбрали такие постановки задач, которые отличаются как по динамике решения, так и по длине интервала. Ожидаемо, в данном случае подобрать оптимальные гиперпараметры оказалось тяжелее из-за возросшей сложности

[1] Анимации доступны по ссылке: https://github.com/DaniilLaptev/PINNs-Training/tree/main/images/animations.

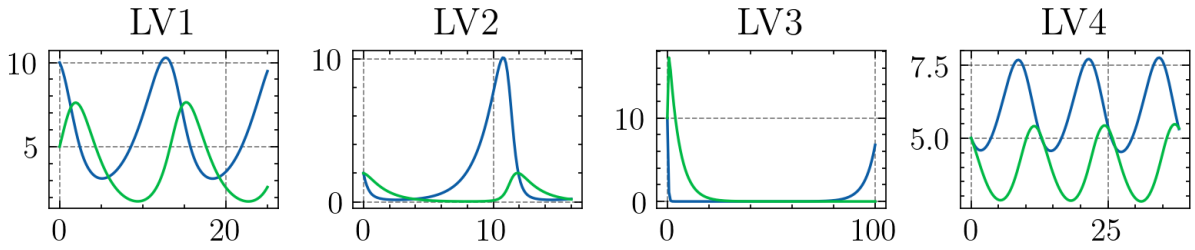

Рис. 4: Различные решения системы Лотки-Вольтерры.

Параметры	LV1	LV2	LV3	LV4
$\alpha, \beta, \delta, \gamma$	0.4, 0.1, 0.1, 0.6	0.7, 1.5, 0.3, 0.7	0.2, 0.2, 0.2, 0.2	0.4, 0.1, 0.1, 0.6
x_0, y_0	10, 5	2, 2	10, 10	5, 5

Таблица 2: Параметры постановки задач для системы (6).

задачи и требуемых вычислительных мощностей. Было взято 100 наборов гиперпараметров для каждой задачи, каждый раз модель обучалась не более чем на 15000 эпохах (алгоритм HPO автоматически прекращал обучение с помощью прунинга, см. секцию B).

Модели с наилучшим результатом представлены на рисунке 5, (a). Видно, что для задачи LV2 модель не смогла обучиться и ушла в плохой режим работы (аппроксимация прямой), а задача LV3 была довольно хорошо приближена в левой части домена, но не была приближена вовсе в правой части. Простоту решения задач можно оценить по рисунку 5, (b) - распределение значений ошибки показывает, что задачи LV2 и LV3 тяжело решить автоматическим подбором гиперпараметров.

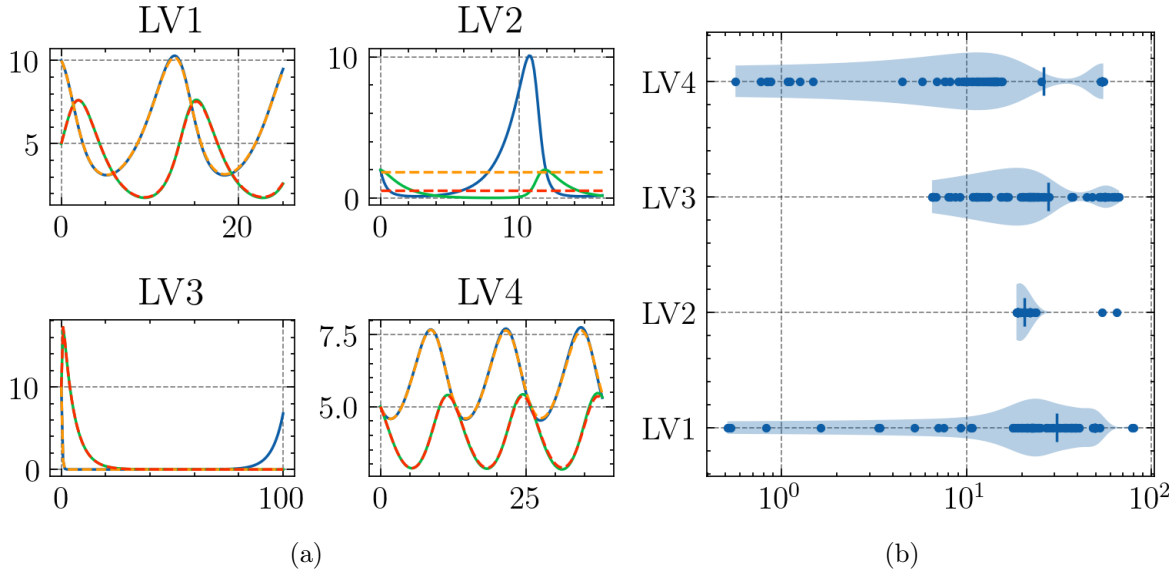

Рис. 5: Результат поиска гиперпараметров посредством алгоритма HPO. Для каждой задачи было выбрано 100 итераций поиска. (a) Предсказания моделей с наименьшей ошибкой, (b) распределение L2 ошибки для каждой задачи.

Мы делаем следующие предположения. В случае LV3 решение в правой части не было приближено потому, что искомая функция практически на всём домене является нулевой, из-за чего модель имеет недостаточно информации о динамике искомой функции в области ближе к концу домена. В случае LV2 плохой режим работы связан в первую очередь с тем, что оптимизатор по каким-то причинам пришёл к плохим весам модели. Иными словами,

мы предполагаем, что причины такого поведения на этих двух задачах различны. Это предположение необходимо также дополнительно проанализировать.

При наблюдении за процессом обучения некоторых моделей нами были выявлены следующие проблемы.

Во-первых, модель зачастую на первых итерациях начинает аппроксимировать константную функцию, как в случае решения задачи LV2 на рисунке 5, (a). Если модель ушла в плохой режим, то практически на всём домене приближается константная функция, и чаще всего - нулевая.

Во-вторых, в данной задаче вновь было обнаружено обучение слева направо. Если модель остаётся в хорошем режиме, то на первых итерациях обучения она в левой части домена приближает форму искомой функции и аппроксимирует начальные условия, а на правой части домена - прямую, причём такую, которая является средним значением искомой функции. Через некоторое число итераций эта прямая начинает изгибаться, и форма функции решения начинает приближаться в тех областях домена, в которых наблюдался изгиб. В зависимости от того, как выглядит искомая функция, таких моментов сгибания может быть несколько.

В-третьих мы уделили больше внимания задаче LV4 и кроме описанных выше проблем обнаружили феномен, который мы назвали двойным обучением [2]. Типичный ход обучения нейросети для неё можно описать следующим образом. На протяжении некоторого числа итераций (в нашем случае это от 6 до 11 тысяч) модель постепенно учится слева направо, постепенно аппроксимируя динамику. В какой-то момент оптимизатор выскочил в область с плохими параметрами, значительно увеличив ошибку. Форма нейросети при этом стала такой, какая была на одной из ранних итераций, то есть оптимизатор как бы вернулся на более ранний этап обучения - например, на 10000 итерации модель могла приобрести форму, близкую к той, которую она имела на 5000 эпохе. После этого оптимизатор обучается как будто бы заново. Иногда он может улучшить результат - например, если на итерации 8500 произошёл сброс, то есть шанс, что на 12000 итерации модель будет лучше приближать искомую функцию, чем на 8000 итерации, однако это происходит не всегда. Такие скачки могут происходить неоднократно. Их частота, как и само возникновение такого феномена, мы связываем с начальной инициализацией модели, удачности шагов оптимизации и выбранного lr, однако пока мы не смогли установить точных причин и связей. Предположительно, здесь есть связь с затухающими градиентами и попыткой оптимизатора выскочить из области локального минимума, из-за чего шаг оптимизации становится достаточно большим, но направление шага оказывается неудачным. Частично это предположение подтверждается визуализацией динамики шага на рисунке 6.

В-четвёртых, добавление в нейросетевую архитектуру слоя Layer Normalization позволило добиться стабильного обучения. Возможно, это позволяет нейросетевой модели лучше работать с различным масштабом входных значений.

Рассмотрим теперь феномен двойного обучения. На рисунке 6 изображён сводный график динамики следующих величин во время обучения на задачу LV4: L2-нормы градиентов параметров для каждого слоя, L2-нормы параметров для каждого слоя, значения функции ошибки и её термов, а также значение L2 ошибки и норма вектора, на который изменились параметры при обучении. Расчёт величин происходит следующим образом. Перед началом обучения сохраняются инициализированные веса θ_0. На итерации $i \in [0, 500, 1000, ..., 15000]$ все величины, кроме шага, рассчитываются для параметров θ_i [3].

[2] Визуализацию можно посмотреть по следующей ссылке: https://github.com/DaniilLaptev/PINNs-Training/tree/main/images/animations/lv4_double_learning.gif. Здесь же можно увидеть феномен обучения слева направо.

[3] Для подробностей процесса вычисления обратитесь к исходному коду: https://github.com/DaniilLaptev/PINNs-Training/blob/main/code/lotkavolterra.ipynb, а также к коду построению графика.

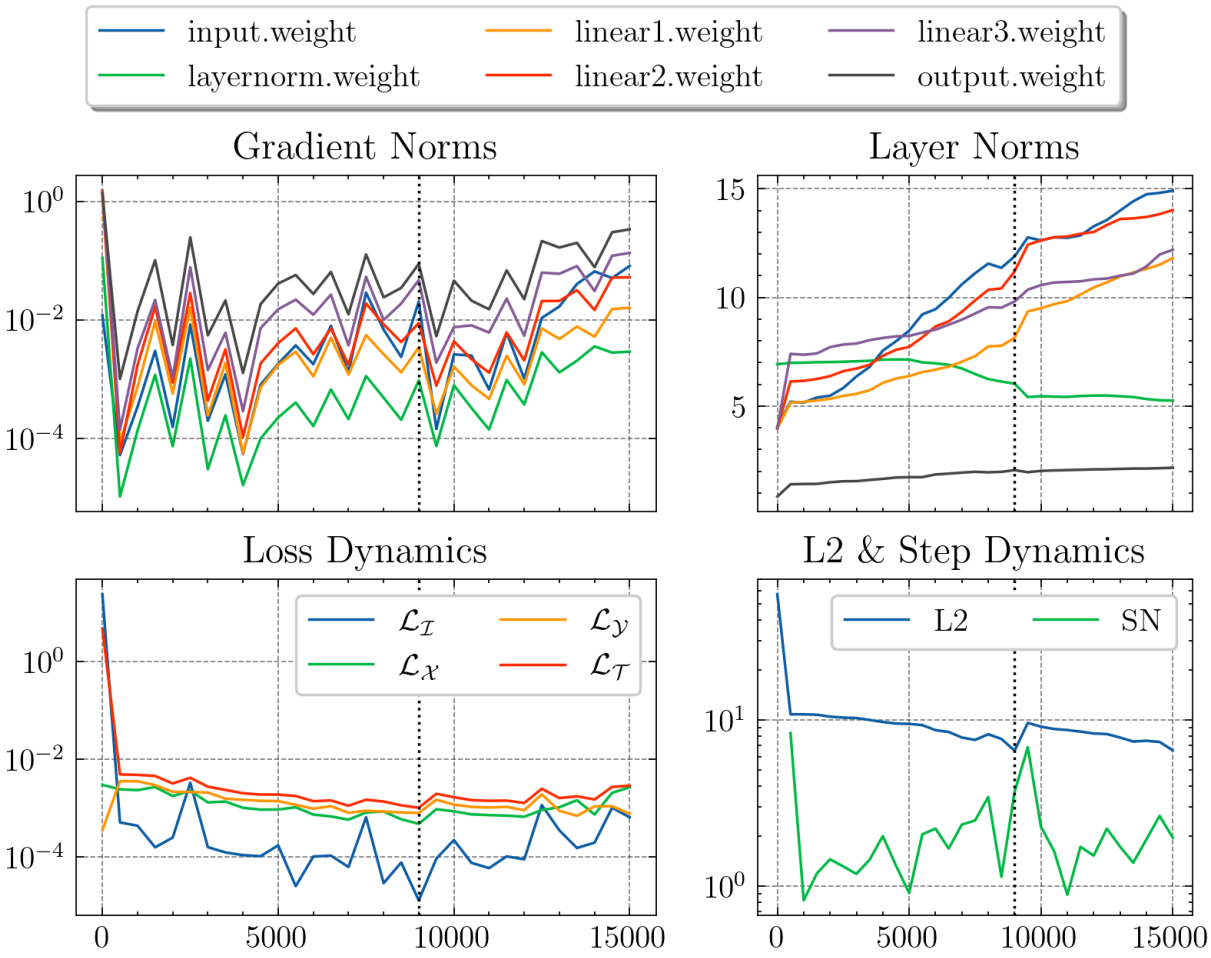

Рис. 6: Сводный график динамики норм градиентов весов, норм самих весов, а также функций потерь и ошибки. Здесь на итерации 10000 (вертикальная точечная линия) изображены: значение функции потерь и L2 ошибки для весов θ_{9000}, значение норм весов и градиентов параметров θ_{9000}, и Step Norm (SN) $= \|\theta_{9000} - \theta_{8500}\|_2$.

Норма шага для итерации i - это величина $\|\theta_{i-1} - \theta_i\|_2$ для всех i кроме нуля.

Видно, что в процессе обучения постепенно повышается норма весов каждого слоя, кроме Layer Normalization. Это нормальное поведение - оно наблюдалось всегда. Можно увидеть резкий рост нормы весов с итерации 9000 до 9500, который сопровождается большой нормой шага оптимизации. Это свидетельствует о том, что оптимизатор принял решение сделать длинный шаг в сторону той области, в которой норма параметров будет большой. Что заставило оптимизатор так далеко отойти в сторону плохого решения? Возможной причиной этому является попытка выбраться из некоторого локального минимума или преодолеть седловую точку. Чтобы сказать что-либо ещё о характере этого поведения оптимизатора, нам необходимо произвести более подробное логирование динамики обучения и собрать дополнительную информацию, а также проанализировать ландшафт функции потерь и движение оптимизатора по этому ландшафту. Также видно, что нормы градиентов постепенно увеличиваются - это может быть связано с постепенным уменьшением нормы весов Layer Normalization, который перестаёт контролировать величину входных значений на первый скрытый слой. Возможно, феномен двойного обучения связан именно с поведением этого слоя. К сожалению, о причинно-следственных связях говорить пока рано.

Это может оказаться довольно обычным поведением нейросетевой модели, однако здесь нас интересует то, что оптимизатор не возвращается на приемлемый уровень качества, а

сбрасывается на одну из ранних итераций и продолжает обучаться от неё (тоже справа налево). Мы подозреваем в первую очередь застревание в локальном минимуме и попытку оптимизатора выбраться из него, которая оказывается в некотором смысле неудачной. Кроме того, мы используем $\mathrm{lr} = 0.01$, что может оказаться достаточно большим значением, и из-за чего могут возникать такие выскакивания оптимизатора. С другой стороны, использование меньшего lr приводило к тому, что нейросетевая модель практически сразу входила в режим плохой аппроксимации. Имеет смысл рассмотреть изменяющийся во времени lr или изменяющееся количество точек коллокации. Также необходим анализ ландшафта потерь и динамики оптимизации [17].

Из методов сэмплирования точек коллокации мы использовали только равномерное разбиение на сетку, поскольку практически во всех задачах этот метод показал себя наилучшим. Можно сказать, что сэмплирование по сетке позволяет нейросетевой модели более равномерно приближать форму искомой функции.

2.3 Уравнение диффузии

Эта модель используется как самостоятельное описание процесса диффузии или как особый случай некоторых других моделей. Мы рассмотрим задачу Дирихле для одномерного случая.

Пусть имеется функция $u(\mathbf{x}, t) : \mathbb{R}^{N+1} \to \mathbb{R}$, которая описывает, например, концентрацию вещества в какой-либо точке пространства и времени (в нашем случае $N = 1$). Пусть число D, называемое коэффициентом диффузии, является постоянным на всей ограниченной области $[A, B] \times [0, T]$. Кроме того, на границах этой области нам известно поведение искомой функции. Тогда краевую задачу для уравнения диффузии мы запишем как

$$\begin{aligned} \frac{\partial u}{\partial t} &= D\frac{\partial^2 u}{\partial x^2}, \\ u(x, 0) &= f(x), \\ u(A, t) &= g_1(t), \\ u(B, t) &= g_2(t), \end{aligned} \tag{8}$$

причём вместо явно заданных функций на границах мы можем иметь лишь некоторый набор измерений. Методы машинного обучения позволят приблизить решение на основе неполных данных, что является ещё одним достоинством PINNs.

Пусть при $t = 0$ и на границах выбирается соответственно N_I, N_A, N_B точек вместе с заданными в них значениями искомой функции. Внутри области мы берём N_D произвольных точек. Тогда функции потерь мы можем записать как

$$\begin{aligned} L_{\mathcal{I}} &= \frac{1}{N_I}\sum_{i=1}^{N_I}(\mathcal{N}(x_i, 0) - f(x_i))^2, \\ L_{\mathcal{B}} &= \frac{1}{N_A}\sum_{i=1}^{N_A}(\mathcal{N}(A, t_i) - g_1(t_i))^2 + \frac{1}{N_B}\sum_{i=1}^{N_B}(\mathcal{N}(B, t_i) - g_2(t_i))^2, \\ L_{\mathcal{D}} &= \frac{1}{N_D}\sum_{i=1}^{N_D}\left(\frac{\partial \mathcal{N}}{\partial t}(x_i, t_i) - D\frac{\partial^2 \mathcal{N}}{\partial x^2}(x_i, t_i)\right)^2 \end{aligned} \tag{9}$$

Полная функция потерь в нашей реализации выглядит следующим образом:

$$L = C(L_{\mathcal{I}} + L_{\mathcal{B}}) + (1 - C)L_{\mathcal{D}}$$

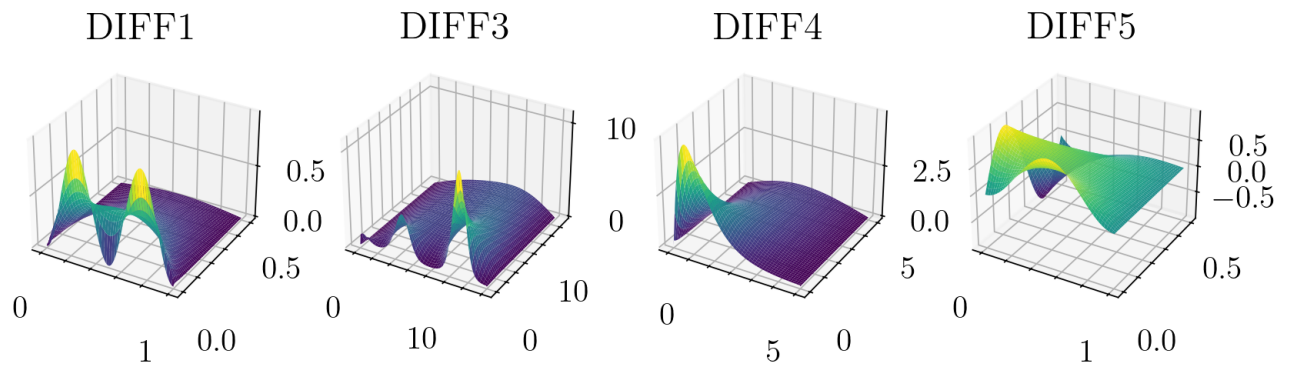

Рис. 7: Рассматриваемые решения уравнения диффузии. DIFF2 был пропущен, поскольку он визуально неотличим от DIFF1.

Параметры	DIFF1	DIFF2	DIFF3	DIFF4	DIFF5
$f(x)$	$\sin^2(2\pi x)$	$\sin^2(\frac{\pi t}{2})$	$(\frac{1}{x+\epsilon})^{\sin(x)}$	$5\sin(e^{-x+0.145})$	$\sin(4\pi t)$
$g_1(t)$, $g_2(t)$	0, 0	0, 0	0, 0	0, 0	$\sin^2(\pi t)$, 0
D	0.5	0.1	0.4	0.7	0.5
A, B, T	0, 1, 0.5	-2, 2, 40	0, 14, 15	-1, 5, 5	0, 1, 0.5

Таблица 3: Параметры постановки задач для системы 8. ϵ означает машинный эпсилон, который не даёт знаменателю стать нулём.

На рисунке 7 представлены решения, которые мы аппроксимируем с помощью PINNs. В таблице 3 представлены параметры постановки задач. Здесь вновь использовался автоматический поиск гиперпараметров, по 50 итераций поиска для каждой задачи. Результат оказался более успешным, чем в случае системы Лотки-Волтерры - все задачи удалось обучить на приемлемом уровне без каких-либо трудностей.

Были сделаны следующие выводы.

Во-первых, визуализация процесса обучения, а также графиков функции потерь показывает, что при обучении нейросетевая модель также проходит несколько фаз. Для задачи DIFF1 эти фазы соответствуют минимизации различных термов функции потерь, причём сначала граничных условий, потом внутренних. Кроме того, первая фаза аппроксимации похожа на то, как если бы решение в нуле имело форму $\sin(2\pi t)$ с одной вершиной (пик концентрации), а затем происходит разделение на две вершины. Этот эффект зависит, с одной стороны, от выбранного lr, а с другой стороны - от самой задачи, поскольку для DIFF5, например, такого поведения не наблюдалось.

Во-вторых, мы исследовали влияние метода инициализации на качество решения. Мы не рассматривали Uniform инициализацию, поскольку она ни в одной задаче не показывала себя хорошо. Наилучшими оказались Orthogonal, Xavier Normal и Kaiming Normal. Среди лучших результатов из тех, что нашёл алгоритм HPO, эти методы превалируют.

Явления двойного обучения замечено не было. Вместо него мы обнаружили, что в случае большого lr с течением времени график функций потерь и ошибки становится очень шумным, хоть и постепенно понижается. Это может быть связано как с недостатком информации для модели (можно попробовать рассмотреть изменяющееся во времени число точек коллокации), так и с особенностями ландшафта потерь. В целом, это поведение является нормальным для многих задач глубокого обучения.

Метод сэмплирования точек коллокации мы задали как категориальную переменную в алгоритме HPO, и не смогли выявить каких-либо интересных закономерностей - для каких-то задач наиболее частым среди лучших гиперпараметров оказался метод разбиения по сетке, для каких-то - случайный выбор точек из области.

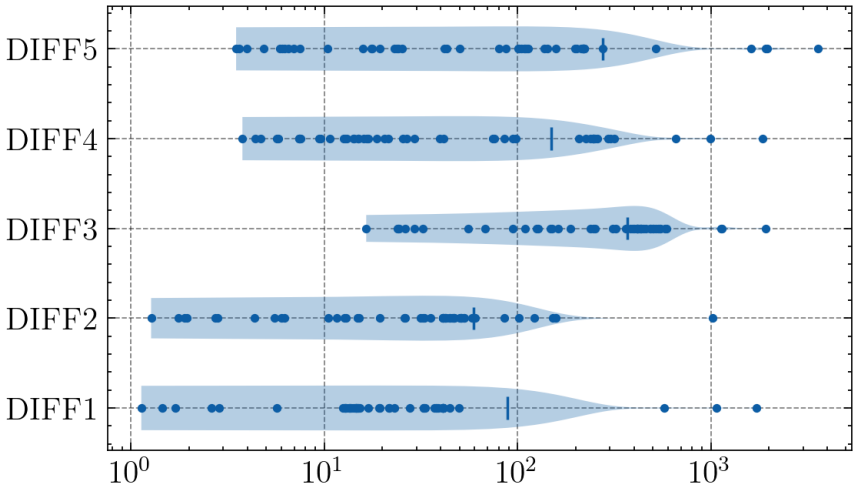

Рис. 8: Распределение ошибок для каждой задачи. Видно, что плохие решения довольно редки, и алгоритм HPO здесь срабатывает так, как от него это ожидается.

В-третьих, эта задача является достаточно простой. Неясно, однако, по каким причинам её решить легче, чем систему Лотки-Вольтерры. Как и в описанных ранее наблюдениях, проблема может заключаться в выборе функции активации. На рисунке 8 изображено распределение значений ошибки для каждой задачи, и на нём явно видно достаточную редкость плохих инициализаций гиперпараметров (которые приводят к плохому режиму аппроксимации или просто сильно отклоняются от численного решения).

2.4 Система Лоренца

2.5 Модель Грея-Скотта

2.6 Выводы раздела

Подводя итоги, мы можем охарактеризовать основные проблемы обучения PINNs, которые нам удалось выявить в результате экспериментов:

1. Нейросетевые модели очень чувствительны к инициализации весов. Эксперименты показали, что от неё в первую очередь зависит успех или неуспех обучения. Для разных задач и разных нейросетевых моделей подходящая инициализация оказалась также различной.

2. Динамика искомой функции значительно влияет на то, как нейросетевая модель обучается и какую информацию она извлекает из функции потерь. Кроме того, выбор функции активации также влияет на ход обучения из-за эффекта затухающего градиента и сатурации (свойства, при котором значение функции активации ограничено асимптотически сверху, как для tanh или сигмоиды), которые мешают нейросетевой модели извлекать информацию при больших значениях входного аргумента.

3. В силу особенностей формулировки задачи, неясно, что считать переобучением нейросетевой модели. Вместо переобучения, которое обычно выражается в ухудшении качества работы модели после какого-то этапа обучения, мы, наоборот, сталкиваемся с трудностями получения всё более низкого значения функции потерь. Качество может ухудшиться, но только вместе с увеличением значения функции потерь на «тренировочном» наборе данных (граничные условия + точки коллокации). Это

подтверждает наше предположение о том, что если задача поставлена корректно, то нейросетевая модель сойдётся к её решению.

4. В некоторых случаях ухудшение качества работы нейросетевой модели выражалось в том, что в один момент она как бы сбрасывалась на более ранний этап обучения, принимала такую форму, которую уже имела раньше. Однако во многих случаях после этого обучение продолжалось как будто бы заново, и мы могли достигать всё более низких значений функции потерь и ошибок, притом, что нормы каждого слоя нейросетевой модели практически монотонно изменялись. Этот эффект мы назвали двойным обучением.

5. Пронаблюдав за динамикой обучения моделей на задачи гармонического осциллятора и системы Лотки-Вольтерры, мы выявили эффект, который мы характеризуем как обучение слева направо Нейросетевая модель сначала повторяет форму искомой функции на левой части домена, затем на правой части домена. Это скорее всего связано с наблюдениями 2. и 4., т.е. с затухающими градиентами и сатурацией, а также с тем, что нейросетевая модель знает значение функции лишь в нуле, а всю остальную динамику ей приходится учить, опираясь на эту информацию (и таким образом можно сказать, что нейросеть аппроксимирует информацию справа, основываясь на информации слева, используя выученную динамику слева как опору для дальнейшего обучения).

Далее мы будем исследовать причины возникновения описанных феноменов и изучать гиперпараметры.

3 Анализ гиперпараметров и процесса обучения

Мы занимаемся поиском точки θ^* из пространства параметров $\mathbb{R}^p$, которая минимизирует (3). Поиск производится итеративно с помощью алгоритма градиентного спуска. На i-й итерации для набора данных $\mathbf{X}$ мы вычисляем градиент функции потерь $\nabla\mathcal{L}(\mathbf{X};\theta_i)$ относительно параметров θ_i с помощью автоматического дифференцирования, и выполняем переход к θ_{i+1} согласно какому-либо правилу (алгоритму оптимизации), тем самым обновляя параметры нашей нейросетевой модели.

Этот процесс можно представить как движение оптимизатора по *ландшафту потерь*, который определяется как поверхность, заданная сильно невыпуклой функцией потерь $\mathcal{L}:\mathbb{R}^p\to\mathbb{R}$. В силу невыпуклости ландшафта, а также неоптимальной желаемой скорости сходимости оптимизатора, мы можем столкнуться с различными проблемами его движения (например, застревание в локальных минимумах или резкие минимумы, в которые оптимизатор зайти не может), которые обычно преодолеваются либо конструированием нового ландшафта потерь, либо подбором более подходящих гиперпараметров.

Таким образом, в вопросе сходимости процесса обучения является важным рассмотрение геометрии ландшафта потерь и движения по нему оптимизатора. Теоретические исследования показывают, что геометрия ландшафта потерь имеет некоторую связь со способностью нейросетевой модели к обобщению, и, кроме того, выбор нейросетевой архитектуры может значительно влиять на сложность оптимизации из-за изменения ландшафта потерь [11].

Введём следующую классификацию параметров:

1. Краевая задача: дифференциальное уравнение, граничные условия, параметры γ.

2. Нейросетевая модель: архитектура и конкретный способ её использования.

3. Задача оптимизации: функция потерь и коэффициенты входящих в неё термов.

4. Обучение: параметры оптимизатора, инициализация модели, точки коллокации.

4 Теоретические гарантии

4.1 Теоретические гарантии сходимости

4.2 Оценка глобальной погрешности решения

Простейший способ проанализировать, насколько хорошо работает наше нейросетевое решение - сравнить его с общепринятыми методами численного решения, а ещё лучше - рассмотреть задачи с доступным точным решением. Доверие к нему, таким образом, будет опираться на опыт. Однако существует ряд теоретических методов, которые мы рассмотрим далее.

Возьмём нотацию, описанную в [3]. Пусть $u(z) : \Omega \to X$ - искомая функция, $\mathcal{N}(z;\theta)$ - нейросетевая модель с фиксированной архитектурой. Мы хотим получить оценку величины

$$\|\mathcal{N}(z;\theta) - u(z)\|,$$

основываясь на информации об архитектуре модели и тренировочных данных, а также, может быть, на догадках о каких-либо свойствах искомой функции.

Стоит ещё раз заметить, что функция $\mathcal{N}(z;\theta)$ не определяется единственным образом своими параметрами, а имеет также и архитектуру - конкретный способ реализации параметров. Он может заметно влиять на качество решения и его также необходимо брать в расчёт. Для того, чтобы стандартизовать анализ, можно ввести понятие сложности функции, или её экспрессивности, и выработать способ количественного измерения этой характеристики. Возможно, это позволит упростить методологию.

Пусть дан набор точек z_i и соответствующих значений функции $u(z_i)$, где $i \in \{1, ..., N\}$ (например, это какая-либо совокупность измерений искомого решения ДУ, начальных и граничных условий). Тогда *эмпирический риск* для модели с фиксированными параметрами θ определяется как

$$\hat{\mathcal{R}}[\mathcal{N}] = \frac{1}{N}\sum_{i=1}^{N} \|\mathcal{N}(z_i) - u(z_i)\|^2$$

Общий *риск* модели, характеризующий её способность аппроксимировать функцию на всём Ω, будет определяться как

$$\mathcal{R}[\mathcal{N}] = \int_{\Omega} (\mathcal{N}(z) - u(z))^2 dz$$

Качество решения можно оценить, исходя из трёх факторов.

Ошибка оптимизации. Обозначим модель, полученную в результате тренировки, как $\mathcal{N}^*$. Мы введём *ошибку оптимизации* как меру качества тренировочного процесса для данной нейросетевой архитектуры на данных тренировочных точках. Мы ожидаем, что если достигнут минимум ошибки оптимизации, дальнейший способ улучшения качества - поиск новых данных и использование новых архитектур.

Обычно ошибка оптимизации нам неизвестна на практике, т.к. функция потерь является сильно невыпуклой, а стандартные методы оптимизации дают лишь приближённое решение (нередко застревая в локальном минимуме). Математически она будет выражаться

как разница между эмпирическим риском полученной модели $\hat{u}^*$ и наименьшим эмпирическим риском из всех возможных:

$$\varepsilon_O = \hat{\mathcal{R}}[\mathcal{N}^*] - \inf_\theta \hat{\mathcal{R}}[\mathcal{N}]$$

Равенство $\varepsilon_O = 0$ будет означать, что мы нашли глобальный минимум.

Ошибка аппроксимации. Введём *ошибку аппроксимации*, описывающую, насколько хорошо мы можем аппроксимировать искомую функцию, используя данную архитектуру. Математически она определяется как наименьший возможный общий риск для данной архитектуры:

$$\varepsilon_A = \inf_\theta \mathcal{R}[\mathcal{N}]$$

Способность нейросетей аппроксимировать различные классы функций широко изучена в литературе и именно в данном контексте играют роль теоремы универсальной аппроксимации, которые, в сущности, утверждают, что ϵ_A можно сделать сколь угодно малой при правильном выборе архитектуры модели и количества параметров. Редко, однако, говорится о том, как именно выбирать архитектуру и размер модели - эта область ещё мало изучена, хоть в ней и есть свои результаты.

Ошибка обобщения. Она характеризует способность нейросети делать верные предсказания на тех данных, которые не встречались в тренировочном наборе. Она выразится как максимальное отклонение общего риска от эмпирического риска при данной архитектуре и данных тренировочных точках:

$$\varepsilon_G = \sup_\theta |\mathcal{R}[\mathcal{N}] - \hat{\mathcal{R}}[\mathcal{N}]|$$

Заключение

A Методы получения численного решения дифференциальных уравнений

B Метод подбора гиперпараметров и метод анализа их роли

Список источников

[1] Shengze Cai и др. *Physics-informed neural networks (PINNs) for fluid mechanics: A review*. 2021. arXiv: 2105.09506 [physics.flu-dyn].

[2] Hugh Bishop Christopher M. Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024.

[3] Salvatore Cuomo и др. «Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next». В: *Journal of Scientific Computing* 92.3 (июль 2022), с. 88. ISSN: 1573-7691. DOI: 10.1007/s10915-022-01939-z. URL: https://doi.org/10.1007/s10915-022-01939-z.

[4] Jan E. Gerken и др. «Geometric deep learning and equivariant neural networks». В: *Artificial Intelligence Review* 56.12 (дек. 2023), с. 14605—14662. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10502-7. URL: https://doi.org/10.1007/s10462-023-10502-7.

[5] Zhongkai Hao и др. «Physics-informed machine learning: A survey on problems, methods and applications». В: *arXiv preprint arXiv:2211.08064* (2022).

[6] Zhongkai Hao и др. «PINNacle: A Comprehensive Benchmark of Physics-Informed Neural Networks for Solving PDEs». В: *arXiv preprint arXiv:2306.08827* (2023).

[7] Ruimeng Hu и др. *Higher-Order error estimates for physics-informed neural networks approximating the primitive equations*. 2023. arXiv: 2209.11929 [math.NA].

[8] Ameya D Jagtap и George Em Karniadakis. «Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations». В: *Communications in Computational Physics* 28.5 (2020), с. 2002—2041.

[9] Ameya D Jagtap, Ehsan Kharazmi и George Em Karniadakis. «Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems». В: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), с. 113028.

[10] Anastasis Kratsios. «The Universal Approximation Property». В: *Annals of Mathematics and Artificial Intelligence* 89.5 (июнь 2021), с. 435—469. ISSN: 1573-7470. DOI: 10.1007/s10472-020-09723-1. URL: https://doi.org/10.1007/s10472-020-09723-1.

[11] Hao Li и др. *Visualizing the Loss Landscape of Neural Nets*. 2018. arXiv: 1712.09913 [cs.LG].

[12] Jie-Ying Li и др. «Utilizing symmetry-enhanced physics-informed neural network to obtain the solution beyond sampling domain for partial differential equations». В: *Nonlinear Dynamics* 111.23 (дек. 2023), с. 21861—21876. ISSN: 1573-269X. DOI: 10.1007/s11071-023-08975-w. URL: https://doi.org/10.1007/s11071-023-08975-w.

[13] Remco van der Meer, Cornelis Oosterlee и Anastasia Borovykh. *Optimally weighted loss functions for solving PDEs with Neural Networks*. 2021. arXiv: 2002.06269 [math.NA].

[14] Siddhartha Mishra и Roberto Molinaro. «Estimates on the generalization error of physics-informed neural networks for approximating PDEs». В: *IMA Journal of Numerical Analysis* 43.1 (янв. 2022), с. 1—43. ISSN: 0272-4979. DOI: 10.1093/imanum/drab093. eprint: https://academic.oup.com/imajna/article-pdf/43/1/1/49059512/drab093.pdf. URL: https://doi.org/10.1093/imanum/drab093.

[15] Maithra Raghu и Eric Schmidt. *A Survey of Deep Learning for Scientific Discovery*. 2020. arXiv: 2003.11755 [cs.LG].

[16] M. Raissi, P. Perdikaris и G.E. Karniadakis. «Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations». В: *Journal of Computational Physics* 378 (2019), 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.

[17] Pratik Rathore и др. *Challenges in Training PINNs: A Loss Landscape Perspective*. 2024. arXiv: 2402.01868 [cs.LG].

[18] Franz M. Rohrhofer, Stefan Posch и Bernhard C. Geiger. *On the Pareto Front of Physics-Informed Neural Networks*. 2021. arXiv: 2105.00862 [cs.LG].

[19] Tim De Ryck, Ameya D. Jagtap и Siddhartha Mishra. *Error estimates for physics informed neural networks approximating the Navier-Stokes equations*. 2023. arXiv: 2203.09346 [math.NA].

[20] Khemraj Shukla, Ameya D Jagtap и George Em Karniadakis. «Parallel Physics-Informed Neural Networks via Domain Decomposition». В: *Journal of Computational Physics* 447 (2021), с. 110683.