КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2024 Т. 16 № 7 С. 1621–1636
DOI: 10.20537/2076-7633-2024-16-7-1621-1636

Ки&М

СПЕЦИАЛЬНЫЙ ВЫПУСК

УДК: 519.622

Особенности применения физически информированных нейронных сетей для решения обыкновенных дифференциальных уравнений

И. В. Конюхов[1,2,3,a], В. М. Конюхов[4,b], А. А. Черница[1,c], А. Дюсенова[1,d]

[1]Университет Иннополис,
Россия, 420500, г. Иннополис, ул. Университетская, д. 1
[2]КазО МСЦ РАН – филиал ФГУ ФНЦ НИИСИ РАН,
Россия, 420111, г. Казань, ул. Лобачевского, д. 2, корп. 31
[3]Национальный исследовательский центр «Курчатовский институт»,
Россия, 123098, г. Москва, пл. Академика Курчатова, д. 1
[4]ФГАОУ ВО «Казанский (Приволжский) федеральный университет»,
Россия, 420008, г. Казань, ул. Кремлевская, д. 18, корп. 1

E-mail: [a] i.konyukhov@innopolis.ru, [b] vladimir.konyukhov@kpfu.ru, [c] a.chernitsa@innopolis.university, [d] a.dyusenova@innopolis.university

Получено 22.10.2024, после доработки — 12.11.2024
Принято к публикации 25.11.2024

Рассматривается применение физически информированных нейронных сетей с использованием многослойных персептронов для решения задач Коши, в которых правые части уравнения являются непрерывными монотонно возрастающими, убывающими или осциллирующими функциями. С помощью вычислительных экспериментов изучено влияние метода построения приближенного нейросетевого решения, структуры нейронной сети, алгоритмов оптимизации и средств программной реализации на процесс обучения и точность полученного решения. Выполнен анализ эффективности работы наиболее часто используемых библиотек машинного обучения при разработке программ на языках программирования Python и C#. Показано, что применение языка C# позволяет сократить время обучения нейросетей на 20–40 %. Выбор различных функций активации влияет на процесс обучения и точность приближенного решения. Наиболее эффективными в рассматриваемых задачах являются сигмоида и гиперболический тангенс. Минимум функции потерь достигается при определенном количестве нейронов скрытого слоя однослойной нейронной сети за фиксированное время обучения нейросетевой модели, причем усложнение структуры сети за счет увеличения числа нейронов не приводит к улучшению результатов обучения. При этом величина шага сетки между точками обучающей выборки, обеспечивающей минимум функции потерь, в рассмотренных задачах Коши практически одинакова. Кроме того, при обучении однослойных нейронных сетей наиболее эффективными для решения задач оптимизации являются метод Adam и его модификации. Дополнительно рассмотрено применение двух- и трехслойных нейронных сетей. Показано, что в этих случаях целесообразно использовать алгоритм LBFGS, который по сравнению с методом Adam в ряде случаев требует на порядок меньшего времени обучения при достижении одинакового порядка точности. Исследованы также особенности обучения нейронной сети в задачах Коши, в которых решение является осциллирующей функцией с монотонно убывающей амплитудой. Для них необходимо строить нейросетевое решение не с постоянными, а с переменными весовыми коэффициентами, что обеспечивает преимущество такого подхода при обучении в тех узлах, которые расположены вблизи конечной точки интервала решения задачи.

Ключевые слова: обыкновенные дифференциальные уравнения, машинное обучение, физически информированные нейронные сети, численные методы

Работа выполнена при (частичной) финансовой поддержке НИЦ «Курчатовский институт»: в рамках Государственного задания (проект № FNEF-2024-0016).

COMPUTER RESEARCH AND MODELING
2024 VOL. 16 NO. 7 P. 1621–1636
DOI: 10.20537/2076-7633-2024-16-7-1621-1636

Ки&М

SPECIAL ISSUE

UDC: 519.622

Analysis of the physics-informed neural network approach to solving ordinary differential equations

I. V. Konyukhov[1,2,3,a], V. M. Konyukhov[4,b], A. A. Chernitsa[1,c], A. Dyussenova[1,d]

[1]Innopolis University,
1 Universitetskaya st., Innopolis, 420500, Russia
[2]KazD JSCC RAS — Branch of SRISA,
2/31 Lobachevskogo st., Kazan, 420111, Russia
[3]Scientific Research Center «Kurchatov Institute»,
1 Akademika Kurchatova pl., Moscow, 123182, Russia
[4]Kazan Federal University,
18/1 Kremlyovskaya st., Kazan, 420008, Russia

E-mail: [a] i.konyukhov@innopolis.ru, [b] vladimir.konyukhov@kpfu.ru, [c] a.chernitsa@innopolis.university, [d] a.dyusenova@innopolis.university

Received 22.10.2024, after completion — 12.11.2024
Accepted for publication 25.11.2024

Considered the application of physics-informed neural networks using multi layer perceptrons to solve Cauchy initial value problems in which the right-hand sides of the equation are continuous monotonically increasing, decreasing or oscillating functions. With the use of the computational experiments the influence of the construction of the approximate neural network solution, neural network structure, optimization algorithm and software implementation means on the learning process and the accuracy of the obtained solution is studied. The analysis of the efficiency of the most frequently used machine learning frameworks in software development with the programming languages Python and C# is carried out. It is shown that the use of C# language allows to reduce the time of neural networks training by 20–40 %. The choice of different activation functions affects the learning process and the accuracy of the approximate solution. The most effective functions in the considered problems are sigmoid and hyperbolic tangent. The minimum of the loss function is achieved at the certain number of neurons of the hidden layer of a single-layer neural network for a fixed training time of the neural network model. It's also mentioned that the complication of the network structure increasing the number of neurons does not improve the training results. At the same time, the size of the grid step between the points of the training sample, providing a minimum of the loss function, is almost the same for the considered Cauchy problems. Training single-layer neural networks, the Adam method and its modifications are the most effective to solve the optimization problems. Additionally, the application of two- and three-layer neural networks is considered. It is shown that in these cases it is reasonable to use the LBFGS algorithm, which, in comparison with the Adam method, in some cases requires much shorter training time achieving the same solution accuracy. The specificity of neural network training for Cauchy problems in which the solution is an oscillating function with monotonically decreasing amplitude is also investigated. For these problems, it is necessary to construct a neural network solution with variable weight coefficient rather than with constant one, which improves the solution in the grid cells located near by the end point of the solution interval.

Keywords: ordinary differential equations, machine learning, physics-informed neural networks, numerical methods

Citation: *Computer Research and Modeling*, 2024, vol. 16, no. 7, pp. 1621–1636 (Russian).

Введение

Математическое описание многих реальных природных и технологических явлений часто включает в себя обыкновенные дифференциальные уравнения (ОДУ). Одним из примеров является квазистационарная модель тепломассопереноса в многоступенчатых центробежных электронасосах и трубах нефтедобывающих скважин, транспортирующих на поверхность земли многофазные (газ – нефть – вода) смеси при разработке нефтяных месторождений. Система дифференциальных уравнений для расчета профилей давления $P(z)$ и температуры $T(z)$ вдоль ступеней насоса и труб скважины может быть записана в виде

$$\begin{cases} \dfrac{dP}{dz} = f_1(P,\, T), & P|_{z=0} = P_0, \\ \dfrac{dT}{dz} = f_2(P,\, T), & T|_{z=0} = T_0, \end{cases}$$

где z — вертикальная пространственная координата; f_1 и f_2 — правые части уравнений, определяются сложным образом на основе зависимостей физико-химических свойств отдельных фаз и смеси (таких как вязкость, плотность, скорость и т. д.), а также полуэмпирических соотношений для расчета различных характеристик потока. Как показано в [Конюхов, 1990], для расчета гидродинамических характеристик (давления, скоростей фаз и смеси и т. д.) можно воспользоваться изотермическим приближением описания процесса при средней температуре T^* смеси в скважине и электронасосе с помощью одного дифференциального уравнения первого порядка:

$$\frac{dP}{dz} = f_1\,(P,\, T^*), \quad P|_{z=0} = P_0.$$

Если ОДУ допускает получение аналитического решения, то искомая функция может быть вычислена при любом значении аргумента. Однако очень часто при решении прикладных задач точное решение ОДУ не может быть найдено. В этом случае приближенные решения строятся с использованием численных методов в ограниченной дискретной области изменения аргумента. При этом расчет значения искомой функции в некоторой произвольной точке интервала требует последовательного вычисления ее значений во всех предыдущих дискретных точках. Однако, например, при итерационном решении задачи оптимизации квазистационарного процесса нефтедобычи подобные расчеты в скважине приходится повторять многократно, вычисляя значение устьевого давления в точке $z = H$ на поверхности земли по заданному забойному давлению P_0 при $z = 0$. Это приводит к значительным вычислительным затратам машинного времени.

Современный метод решения дифференциальных уравнений [Бурнаев и др., 2022], основанный на применении многослойных перцептронов в качестве универсальных аппроксиматоров решения, известный как физически информированные нейронные сети (Physics-Informed Neural Networks), впервые предложен в работе [Lagaris, Likas, Fotiadis, 1998]. Его преимуществом является возможность вычислять значение искомой функции в любой точке области решения без последовательного вычисления ее значений в предыдущих узлах, как и при наличии аналитического решения, без привязки к точкам дискретной сеточной области.

Многослойный перцептрон — тип искусственной нейронной сети, состоящий из нескольких слоев взаимосвязанных нейронов. Эти слои включают входной слой (а), один или несколько скрытых слоев (б) и выходной слой (в) (см. приведенный на рис. 1 пример двухслойного перцептрона).

Каждый нейрон во входном и скрытых слоях соединен с каждым нейроном в последующем слое, образуя полносвязную нейронную сеть. Искусственный нейрон — это фундаментальная единица искусственных нейронных сетей [Petriu, 2004]. Принцип работы отдельного нейрона представлен на рис. 2.

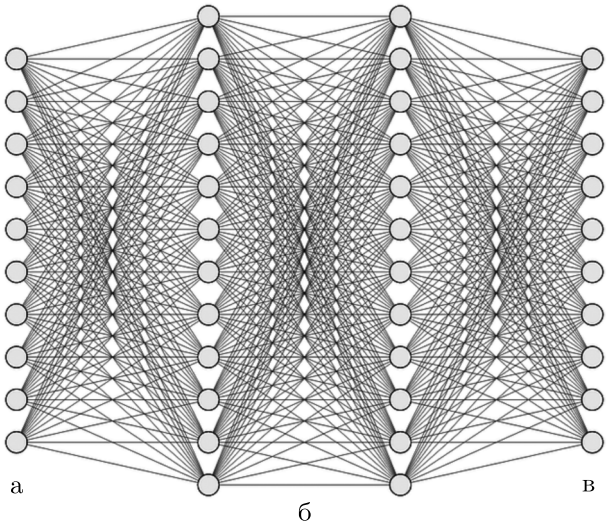

Рис. 1. Многослойный перцептрон

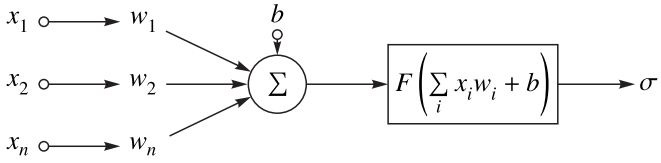

Рис. 2. Принцип работы искусственного нейрона

Нейрон получает один или несколько входных сигналов x_i, обрабатывает их и вырабатывает выходной сигнал σ. Каждый входной сигнал имеет вес w_i, который определяет интенсивность взаимодействия между нейронами. Взвешенные значения $x_i w_i$ входных сигналов суммируются в нейроне $\left(S = \sum_i x_i w_i + b\right)$ и к полученной величине S добавляется некоторое сдвиговое значение b, которое позволяет нейрону иметь ненулевой выходной результат σ даже при нулевых значениях x_i всех входных сигналов. После этого к полученному суммарному значению применяется функция активации $F = F(S + b)$, что формирует конечный выходной результат σ работы отдельного нейрона. Отметим, что при реализации физически информированных нейронных сетей, предназначенных для решения дифференциальных уравнений, для обработки суммарных входных сигналов $S + b$ используются нелинейные функции активации. Благодаря использованию сдвиговых значений и нелинейных функций активации нейронные сети могут моделировать весьма сложные закономерности между входных и выходных данных, что в рассматриваемом случае решения ОДУ позволяет повысить его точность [Shiruru, 2016].

Набор весов w_i и смещений b формирует массив обучаемых параметров нейронной сети. В процессе обучения их значения итеративно корректируются с помощью алгоритмов оптимизации с целью минимизации ошибки между предсказаниями сети и реальными результатами. Эффективность искусственной нейронной сети во многом зависит от правильного выбора этих обучаемых параметров.

Рассмотрим математическую формулировку данного подхода на примере задачи Коши с начальным условием, записанной в виде

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0, \tag{1}$$

где y — неизвестная функция, t — ее аргумент, y_0 — заданное значение функции y в точке t_0.

Будем искать решение задачи (1) с использованием нейронной сети на ограниченном интервале $[t_0, t_n]$. Если представить результат ее работы как вычисление функции $\Upsilon(t, \vec{p})$, аппроксимирующей решение $y(t)$ при всех значениях аргумента $t \in [t_0, t_n]$ и определяемой набором $\vec{p}$ ее обучаемых параметров [Hornik, Stinchcombe, White, 1989], то производная Υ'_t должна по построению аппроксимировать производную $y'_t(t)$, заданную правой частью уравнения (1), а именно $\Upsilon'_t(t, \vec{p}) \approx f(t, \Upsilon(t, \vec{p}))$ для всех $t \in [t_0, t_n]$ [Lee, Kang, 1990; Lagaris, Likas, Fotiadis, 1998].

Поскольку решение y уравнения (1) неизвестно, то для обучения нейронной сети необходимо использовать его известную правую часть $f(t, y)$ и начальное условие для определения функции потерь L за счет варьирования набора параметров $\vec{p}$. В разных работах [Lee, Kang, 1990; Lagaris, Likas, Fotiadis, 1998; Flamant, Pavlos, Sondak, 2020] авторы приводят разные способы построения функции потерь. Процедура обучения нейронной сети не требует специальных эмпирических данных. В качестве обучающего набора данных при определении функции потерь L можно использовать набор $\vec{\tau}$ точек, равномерно распределенных по интервалу $[t_0, t_n]$. В процессе обучения необходимо минимизировать значение функции потерь в этих точках. Анализ особенностей построения приближенного решения и функции потерь обсуждается далее по ходу изложения полученных результатов для различных видов правой части уравнения (1).

Таким образом, нахождение приближенного решения задачи Коши (1) с помощью нейронной сети сводится к решению задачи оптимизации

$$L(\vec{\tau}, \vec{p}) \underset{\vec{p}}{\to} \min. \tag{2}$$

Для оценки эффективности данного подхода необходимо оценить величину погрешности между приближенным нейросетевым $\widetilde{y}(t) = \Upsilon(t, \vec{p})$ и точным аналитическим $y(t)$ решениями. Так, если обозначить через $\widetilde{e}(t) = \widetilde{y}(t) - y(t)$ и $\widetilde{\varepsilon}(t) = \frac{d\widetilde{y}}{dt} - f(t, \widetilde{y})$, то, полагая, что функция f удовлетворяет условию $|f(t, y(t) + \widetilde{e}(t)) - f(t, y(t))| \leqslant C_L|\widetilde{e}(t)|$ непрерывности по Липшицу, в работе [Flamant, Pavlos, Sondak, 2020] получена следующая оценка:

$$|\widetilde{e}(t)| \leqslant \frac{\widetilde{\varepsilon}_{\max}}{C_L}\left[e^{C_L(t-t_0)} - 1\right], \quad \widetilde{\varepsilon}_{\max} = \max_{t_0 \leqslant t \leqslant t_n} |\widetilde{\varepsilon}(t)|, \tag{3}$$

где C_L — константа Липшица. Отметим, что условие (3) по форме аналогично оценке невязки $|\overline{e}_i| = |\overline{y}_i - y_i|$ между численным $(\overline{y}_i)$ и аналитическим (y_i) решениями в узлах сетки t_i:

$$|\overline{e}_i| \leqslant \frac{\overline{\varepsilon}_{\max}}{C_L}\left(e^{C_L(t_i - t_0)} - 1\right), \quad \overline{\varepsilon}_{\max} = \max_{0 \leqslant i \leqslant N-1} \left|\frac{\overline{y}_{i+1} - \overline{y}_i}{h} - f(t_i, \overline{y}_i)\right|, \quad i \in 0, 1, \ldots, N,$$

приведенной в [Süli, Mayers, 2003] для явной схемы Эйлера.

При решении сформулированной выше задачи оптимизации (2) важную роль играют методы вычисления производных. В работах разных авторов рассматриваются методы численного (см. [Lee, Kang, 1990; Lagaris, Likas, Fotiadis, 1998; Тюрин, 2019; Семенов, 2024]) и автоматического (см. [Flamant, Pavlos, Sondak, 2020; Gorikhovskii, Evdokimova, Poletansky, 2022]) дифференцирования. Причем алгоритмы автоматического дифференцирования в зависимости от выбранной библиотеки машинного обучения могут быть реализованы различными способами [Baydin et al., 2018].

Целью данной работы является изучение принципов построения приближенных решений ОДУ на основе физически информированных нейронных сетей (включая определение вида решения, функции потерь, размера обучающей выборки данных) и анализ эффективности работы различных библиотек машинного обучения, нейронных сетей различной архитектуры (их функций активации, количеств нейронов, слоев, количественного распределения нейронов по слоям), а также алгоритмов оптимизации.

Методы дифференцирования и средства машинного обучения

Приближенное решение Υ, полученное в результате работы нейронной сети, можно рассматривать как сложную функцию, являющуюся суперпозицией функций активации. В ходе решения задачи оптимизации (2) возникает необходимость в эффективном вычислении производных подобных сложных функций. Существует три основных способа вычисления производных: численный, символьный и алгоритмический (автоматическое дифференцирование) [Baydin et al., 2018].

Численное дифференцирование — это метод приближенного нахождения производных в заданной точке сетки с использованием конечных разностей, определяемых по значениям дискретных функций в соседних точках. Метод полезен в случаях, когда аналитическое дифференцирование затруднительно или невозможно. Однако численное дифференцирование неизбежно сопровождается возникновением ошибок апроксимации и вычислительными погрешностями округления, требуя большого количества вычислений для достижения заданной точности расчетов.

При символьном дифференцировании выражение для производных получается в аналитической форме, позволяя проводить вычисления в любой токе области решения задачи. Однако если символьная формула производной становится сложной, то ее вычисление может быть весьма трудоемким (в том числе — итерационным) процессом.

Автоматическое дифференцирование (АД) отличается от перечисленных выше методов. Его основным преимуществом является способность вычислять производные точно и эффективно, избегая ошибок аппроксимации и округления, характерных для численных методов, и сложности символических выражений. В этом методе сложные функции разбиваются на отдельные элементарные составляющие, к которым применяются обычные правила дифференцирования. Это позволяет вычислять точные производные за время, сравнимое с вычислением значения самой функции. Такое преимущество АД широкое используется в области машинного обучения, компьютерной графики и других областях, где требуются точные и эффективные вычисления производных [Baydin et al., 2018].

В настоящее время наиболее эффективными для обучения нейронных сетей и реализации сложных моделей машинного обучения являются библиотеки TensorFlow и PyTorch, в которых и задействованы методы автоматического дифференцирования, существенно облегчающие вычисления градиентов функции потерь и не требующие ручного вычисления производных. Поскольку в этих библиотеках методы автоматического дифференцирования реализованы несколькими различными способами, описание которых дано, например, в [Paszke et al., 2019], и ориентированы на использование в интерпретируемом языке программирования Python, то представляет интерес сравнение как эффективности реализации физически информированных нейронных сетей с использованием этих библиотеках, так и производительности .NET-версий TensorFlow.NET и TorchSharp этих библиотек при их использовании в связке с компилируемым языком программирования C#.

Библиотеки TensorFlow.NET и TorchSharp позволяют применять различные алгоритмы оптимизации, доступные в исходных библиотеках, такие как методы стохастического градиентного спуска, адаптивной оценки моментов и его вариации. Кроме того, библиотека TorchSharp включает в себя модули, реализующие специальный квазиньютоновский метод оптимизации Бройдена – Флетчера – Гольдфарба – Шанно. Дадим краткую характеристику этих методов, применяющихся для минимизации функции потерь при обучении нейронных сетей [Dami et al., 2019].

Метод стохастического градиентного спуска (Stochastic Gradient Descent, SGD) обновляет параметры модели на каждой итерации, используя случайно выбранное подмножество данных,

в отличие от классического метода градиентного спуска, в котором градиент вычисляется на основе всех данных в обучающем наборе. Такой прием SGD-метода позволяет существенно ускорить процесс обучения, поскольку расчеты на каждой итерации выполняются быстрее и требуют меньше памяти. Достоинством SGD-метода является также то, что он позволяет избегать застревания в локальных минимумах, что особенно важно для сложных и высокоразмерных моделей. Кроме того, благодаря случайным обновлениям метод может более эффективно перебирать набор параметров модели, что способствует нахождению глобального минимума. Однако успешность применения метода существенно зависит от величины шага обучения: слишком большой шаг может привести к расхождению алгоритма, тогда как слишком маленький замедляет процесс обучения.

В алгоритме адаптивного градиента (Adaptive Gradient Algorithm, Adagrad) шаг обучения адаптируется для каждого параметра модели с учетом суммы квадратов предыдущих градиентов. Метод особенно полезен при работе с разреженными данными, где отдельные параметры могут обновляться редко. Основная идея метода Adagrad заключается в том, что параметры, которые редко обновляются, определяются с использованием большего шага обучения, тогда как параметры, обновляемые чаще, — меньшего. Такой подход позволяет более эффективно обучать модель и ускорять процесс сходимости. Недостатком метода Adagrad является то, что шаг обучения может стать очень маленьким при длительном обучении модели, что сильно замедляет процесс сходимости. Для решения этой проблемы предложены другие итерационные алгоритмы, такие как Adadelta и RMSprop (см. [Dami et al., 2019]). Так, в методе Adadelta шаг обучения адаптируется для каждого параметра, но, в отличие от Adagrad, применяется экспоненциальное сглаживание величин градиентов с нескольких предыдущих итераций, что предотвращает излишнее уменьшение шага и делает алгоритм более устойчивым. Основная идея алгоритма RMSProp заключается в автоматическом регулировании шага обучения для каждого параметра в зависимости от частоты его обновлений за счет деления шага обучения на среднюю длину градиента, вычисленного по значениям его модулей с нескольких предыдущих итераций. Подобная нормировка позволяет решить вышеуказанную проблему, особенно в случаях, когда эти модули значительно отличаются по своей величине.

Метод адаптивной оценки моментов (Adaptive Moment Estimation, Adam) объединяет преимущества методов Adagrad и RMSprop. В этом методе также выполняется автоматическая адаптация шага обучения с учетом как среднеарифметических, так и среднеквадратичных значений градиентов для каждого параметра обучающего набора. Такой подход делает алгоритм Adam особенно эффективным и устойчивым при работе с большими и сложными моделями.

Еще один алгоритм LBFGS (Limited-memory Broyden – Fletcher – Goldfarb – Shanno), доступный только в библиотеках машинного обучения PyTorch и TorchSharp, основан на квазиньютоновском методе оптимизации. В нем при вычислении вторых производных используется аппроксимация матрицы Гессе по результатам расчетов только на нескольких предыдущих итерациях (см. [Liu, Nocedal, 1989]). Это позволяет избежать излишнего потребления памяти и обеспечить высокую сходимость, особенно в случаях, когда функции потерь являются гладкими и имеют хорошо обусловленные гессианы. LBFGS особенно полезен для обучения нейронных сетей сложной структуры.

Возможность применения всех алгоритмов будет обсуждаться ниже.

Анализ результатов вычислительных экспериментов

Для исследования производительности нейросетевого метода решения обыкновенных дифференциальных уравнений рассмотрим типичные ситуации, в которых правая часть уравнения (1) является полиномиальной, тригонометрической, экспоненциальной функцией или их

комбинацией:

$$y' = 6t^5, \quad y(0) = 1, \quad t \in [0, 5]; \quad y(t) = t^6 + 1; \tag{4}$$

$$y' = \cos(t), \quad y(0) = 1, \quad t \in [0, 4\pi]; \quad y(t) = \sin(t) + 1; \tag{5}$$

$$y' = -2e^{-2t}, \quad y(0) = 1, \quad t \in [0, 4]; \quad y(t) = e^{-2t}; \tag{6}$$

$$y' = e^{-at}(b\cos(bt) - a\sin(bt)), \quad y(0) = 0, \quad t \in [0, 6\pi]; \quad y(t) = e^{-at}\sin(bt). \tag{7}$$

Здесь $y(t)$ — точные решения соответствующих задач Коши, каждое из которых отличается своим поведением и носит возрастающий (4), убывающий (6) и периодический характер, с постоянной (вариант (5)) либо с убывающей (вариант (7_1) с параметрами $a = 0{,}2$, $b = 2{,}0$) или возрастающей (вариант (7_2) при $a = -0{,}2$, $b = 1{,}2$) амплитудой.

О выборе библиотеки машинного обучения

С целью выбора наиболее эффективных технологий программирования предварительно было выполнено построение тестовой физически информированной нейронной сети с одним скрытым слоем из 40 нейронов с использованием функции активации $F(x) = \text{th}(x)$ и алгоритма оптимизации Adam с 200 равномерно распределенными точками обучающего набора для решения задачи Коши (5) с использованием разных языков программирования и библиотек машинного обучения. Количество M эпох обучения фиксировано и равно 40 000. Отметим, что, как показал анализ специальных вычислительных экспериментов, при $M > 40\,000$ в рассматриваемых задачах не наблюдается увеличение порядка точности нейросетевого решения, но значительно возрастает время обучения.

На ноутбуке с процессором Intel Core Ultra 5 125H процедура обучения с использованием Python и TensorFlow занимает 72 с, Python и PyTorch — 54 с, C# и TensorFlow.NET — 61 с, C# и TorchSharp — 40 с. Видно, что применение языка C# позволяет увеличить производительность процесса обучения на 20–40 %. Аналогичные эксперименты с использованием нейронных сетей с большим количеством слоев и нейронов для решения уравнений с правыми частями более сложного вида показывают такое же ускорение процесса обучения. Поэтому результаты исследований, представленные ниже, приводятся для наиболее производительной комбинации языка C# и библиотеки TorchSharp.

О выборе функции активации

При выборе функций активации $F(x)$ физически информированных нейронных сетей для ОДУ необходимо учитывать нелинейность точного решения $y(t)$. Как отмечается в [Sagar, Simone, Anidhya, 2017], от выбора функции $F(x)$ могут существенно зависеть скорость обучения модели и точность полученного нейросетевого решения.

Библиотеки машинного обучения, в том числе и PyTorch, поддерживают набор разнообразных линейных и нелинейных функций активации. В силу, как правило, нелинейности функции $y(t)$ для построения приближенного нейросетевого решения имеет смысл [Flamant, Pavlos, Sondak, 2020] использовать только нелинейные функции активации с гладкими нелинейными производными первого порядка, например уже реализованные в библиотеке PyTorch функции

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \quad \text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{softplus}(x) = \log(1 + e^x),$$

$$\text{swish}(x) = x \cdot \text{sigmoid}(x), \quad \text{mish}(x) = x \cdot \text{th}(\log(1 + e^x)), \quad \text{gelu}(x) = x \cdot \Phi(x),$$

которые наиболее часто применяются для обучения нейросетей. Здесь $\Phi(x)$ — нормальное интегральное распределение (функция Лапласа).

Для оценки эффективности каждой из этих функций активации при построении приближенного нейросетевого решения задач (4)–(7) были проведены вычислительные эксперименты, в которых параметры нейронной сети задавались так же, как и в предыдущем разделе.

Результаты экспериментов приведены в табл. 1, в которой в верхней строке указаны номера тестовых задач Коши (4)–(7), нижние индексы 1 и 2 соответствуют двум вариантам задачи (7). Для каждого варианта обыкновенного дифференциального уравнения определялись минимальное значение функции потерь L, достигнутое в процессе обучения нейросети в течение 40 000 эпох, и затраченное на обучение время T.

Таблица 1. Результаты расчетов величин функции потерь L и времени T (с) обучения нейросети при решении ОДУ (4)–(7) с использование различных функций активации $F(x)$

$F(x)$	(4)		(5)		(6)		(7_1)		(7_2)	
	L	T	L	T	L	T	L	T	L	T
sigmoid(x)	$\mathbf{1{,}1 \cdot 10^{-3}}$	56	$9{,}1 \cdot 10^{-7}$	76	$7{,}3 \cdot 10^{-8}$	40	$8{,}8 \cdot 10^{-4}$	52	$2{,}1 \cdot 10^{-5}$	61
th(x)	$5{,}9 \cdot 10^{-3}$	60	$\mathbf{3{,}2 \cdot 10^{-7}}$	40	$\mathbf{9{,}8 \cdot 10^{-9}}$	35	$\mathbf{4{,}9 \cdot 10^{-4}}$	51	$\mathbf{1{,}3 \cdot 10^{-5}}$	60
softplus(x)	$7{,}3 \cdot 10^{-3}$	45	$1{,}2 \cdot 10^{-4}$	52	$1{,}3 \cdot 10^{-7}$	45	$3{,}9 \cdot 10^{-3}$	64	$4{,}1 \cdot 10^{-3}$	82
swish(x)	$1{,}6 \cdot 10^{-3}$	41	$4{,}1 \cdot 10^{-4}$	101	$2{,}5 \cdot 10^{-8}$	42	$6{,}3 \cdot 10^{-3}$	61	0,79	65
mish(x)	$1{,}6 \cdot 10^{-3}$	72	$4{,}1 \cdot 10^{-4}$	76	$1{,}4 \cdot 10^{-8}$	72	$7{,}9 \cdot 10^{-3}$	100	0,33	108
gelu(x)	$2{,}2 \cdot 10^{-3}$	74	$1{,}6 \cdot 10^{-5}$	63	$1{,}9 \cdot 10^{-8}$	55	$9{,}5 \cdot 10^{-3}$	82	0,82	102

Как видно из таблицы, в зависимости от выбранной функции активации величина T в каждом варианте ОДУ может отличаться более чем в 2 раза. Отметим, что при фиксированном количестве эпох наилучшая точность L, значение которой в таблице показано жирным шрифтом, не связана с длительностью обучения T и может достигаться при меньших величинах T. Как показали также результаты анализа многочисленных вычислительных экспериментов, выполненных для других типов функциональных зависимостей правой части ОДУ, наиболее эффективными для построения нейросетевого решения являются сигмоида и гиперболический тангенс.

О выборе количества нейронов в скрытом слое

Перейдем теперь к исследованию влияния количества k нейронов в скрытом слое на сходимость нейросетевого решения задач (4)–(7), определяемую величиной функции потерь L, для выбранных функций активации sigmoid(x) и th(x). Вычислительные эксперименты проводились при варьировании величины k и фиксированных прочих исходных данных. Результаты расчетов представлены в табл. 2, которая отличается от табл. 1 тем, что в первом ее столбце приведены значения величины k.

Таблица 2. Расчетные значения величин L и T (с) при решении ОДУ (4)–(7) с использованием однослойных нейронных сетей с различным количеством k нейронов на скрытом слое

k	(4)		(5)		(6)		(7_1)		(7_2)	
	L	T	L	T	L	T	L	T	L	T
10	$1{,}6 \cdot 10^{-2}$	50	$2{,}5 \cdot 10^{-5}$	34	$6{,}2 \cdot 10^{-8}$	33	$6{,}3 \cdot 10^{-3}$	49	—	—
20	$4{,}6 \cdot 10^{-3}$	52	$1{,}4 \cdot 10^{-6}$	37	$6{,}8 \cdot 10^{-8}$	34	$1{,}5 \cdot 10^{-3}$	50	$2{,}1 \cdot 10^{-2}$	57
40	$\mathbf{1{,}1 \cdot 10^{-3}}$	56	$\mathbf{3{,}2 \cdot 10^{-7}}$	40	$\mathbf{9{,}8 \cdot 10^{-9}}$	35	$\mathbf{4{,}9 \cdot 10^{-4}}$	51	$\mathbf{1{,}3 \cdot 10^{-5}}$	60
80	$1{,}6 \cdot 10^{-3}$	71	$3{,}3 \cdot 10^{-6}$	45	$2{,}3 \cdot 10^{-8}$	41	$1{,}1 \cdot 10^{-3}$	61	$4{,}1 \cdot 10^{-4}$	67
160	$1{,}1 \cdot 10^{-3}$	78	$1{,}3 \cdot 10^{-6}$	53	$4{,}6 \cdot 10^{-8}$	55	$8{,}3 \cdot 10^{-4}$	69	$3{,}9 \cdot 10^{-4}$	83

Как и ранее, жирным шрифтом выделены значения, соответствующие наилучшей точности нейросетевого решения, достигнутой при обучении. Приведенные в таблице результаты показы-

вают, что для гладких решений ОДУ в рассматриваемых примерах существует такое количество нейронов скрытого слоя, при котором достигается наименьшее значение функции потерь L за минимальное время обучения T нейросетевой модели. Дальнейшее усложнение структуры нейронной сети за счет увеличения числа k не приводит к существенному улучшению результатов обучения.

О влиянии размера обучающего набора данных

Важным моментом построения нейросетевого решения дифференциального уравнения с требуемой точностью является определение достаточного для этой цели объема данных. В качестве примера на рис. 3 приведены результаты расчетов величины погрешности $E(n) = \max\limits_i |\widetilde{e}_i|$ между точным и нейросетевым решениями уравнения (5) в зависимости от количества узлов сетки n с постоянным шагом разбиения при различных значениях N_D = 20, 25, 30, 200, 480, 640 (кривые 1–6) количества точек набора данных, равномерно распределенных по интервалу $[0, 3\pi]$. Для сравнения показано изменение погрешности численного метода Эйлера $E(n) = \max\limits_i |\overline{e}_i|$ (кривая 7). Вычислительные эксперименты проводились с использованием функции активации $\mathrm{th}(x)$ и алгоритма оптимизации Adam при фиксированном количестве k = 40 нейронов в скрытом слое.

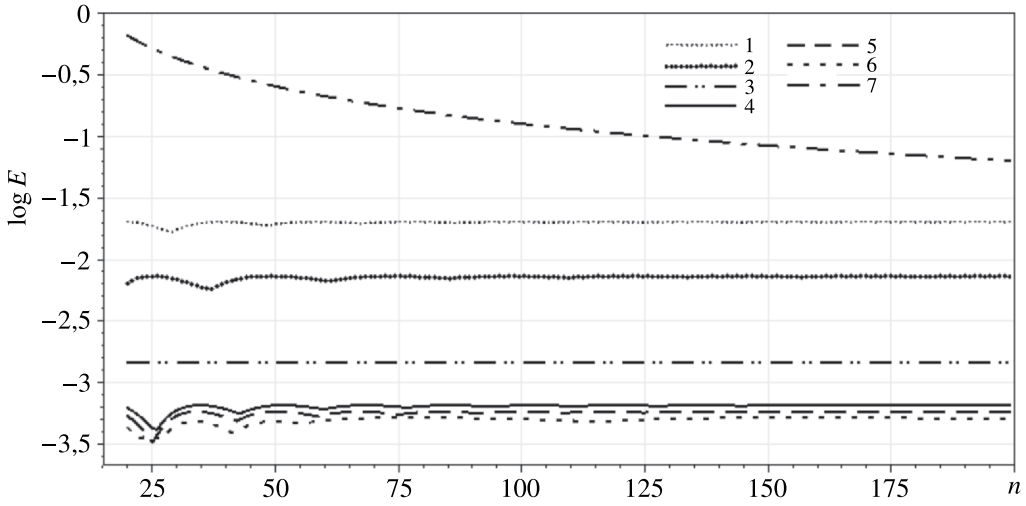

Рис. 3. Зависимость погрешности E нейросетевых (кривые 1–6: N_D = 20, 25, 30, 200, 480, 640) и численного (кривая 7) методов решения ОДУ (5) на интервале $[0, 3\pi]$ от величины n

Как видно на рис. 3, с ростом количества N_D точек набора обучения погрешность E нейросетевого решения немонотонно убывает. Так, при незначительном изменении величины N_D от 20 до 30 (кривые 1–3) погрешность E уменьшается почти на два порядка. Дальнейшее влияние количества N_D по мере его роста до 200 и выше до 640 точек на величину $E(n)$ резко снижается (см. кривые 4–6), причем время T обучения нейронной сети возрастает более чем в два раза. Как показал анализ результатов расчетов, оптимальное количество точек $N_D \approx 200$, что соответствует шагу сетки между точками обучения порядка 0,05. Таким образом, при $N_D > 200$ точность построения нейросетевого решения остается практически неизменной. Отметим также, что для достижения такой же точности численного решения задачи Коши (5) методом Эйлера необходимо использовать сетку с 20 000 точек.

Аналогичные результаты получены и для вариантов ОДУ (4), (6)–(7). В табл. 3 представлены значения параметра N_D, которые являются достаточными для достижения минимальных

значений функций потерь L, выделенных жирным шрифтом в табл. 2. При этом величина шага сетки между точками обучения для всех этих оптимальных случаев также равна ~ 0,05.

Таблица 3. Оптимальное количество точек N_D набора данных для обучения однослойных нейронных сетей одинаковой структуры при решении ОДУ (4)–(7) на интервалах $[t_0, t_n]$

ОДУ	(4)	(5)	(6)	(7_1)	(7_2)
$[t_0, t_n]$	[0, 4]	$[0, 3\pi]$	[0, 5]	$[0, 6\pi]$	$[0, 6\pi]$
N_D	60	200	80	400	400

О выборе алгоритма оптимизации

С целью выбора наиболее подходящего алгоритма обучения нейронных сетей, построенных для решения задач (4)–(7), были проведены многовариантные вычислительные эксперименты с определенными выше наилучшими параметрами нейросетевых моделей: при $k = 40$ (см. табл. 2), со значениями размера N_D обучающей выборки согласно табл. 3, функциями активации sigmoid(x), th(x) (см. табл. 1) и фиксированными прочими исходными данными.

Таблица 4 содержит некоторые основные результаты расчетов с применением описанных выше методов оптимизации, а также алгоритмов AdamW, Adamax и Nadam [Dami et al., 2019], которые являются модификациями метода Adam. Во всех случаях задавались стандартные значения параметров алгоритмов оптимизации (шаг обучения, моменты, весовые коэффициенты и пр.), установленные в библиотеке PyTorch. Прочерки в таблице означают, что при решении рассматриваемых задач Коши некоторые алгоритмы расходятся и нейросеть не может быть обучена. Например, в варианте (4) решение может быть получено только с помощью алгоритмов Adam, AdamW и Adamax.

Таблица 4. Результаты расчетов величин функции потерь L и времени T (с) обучения нейросети при решении ОДУ (4)–(7) с использованием различных алгоритмов оптимизации

Алгоритм	(4)		(5)		(6)		(7_1)		(7_2)	
	L	T	L	T	L	T	L	T	L	T
SGD	—	—	$7{,}5 \cdot 10^{-2}$	32	$1{,}4 \cdot 10^{-4}$	32	$2{,}5 \cdot 10^{-2}$	57	—	—
Adam	$1{,}1 \cdot 10^{-3}$	35	$\mathbf{3{,}2 \cdot 10^{-7}}$	40	$3{,}7 \cdot 10^{-8}$	33	$4{,}9 \cdot 10^{-4}$	54	$1{,}3 \cdot 10^{-5}$	60
AdamW	$2{,}2 \cdot 10^{-3}$	38	$\mathbf{3{,}2 \cdot 10^{-7}}$	40	$\mathbf{9{,}8 \cdot 10^{-9}}$	34	$7{,}9 \cdot 10^{-4}$	62	$1{,}4 \cdot 10^{-5}$	62
Adamax	$\mathbf{7{,}9 \cdot 10^{-4}}$	43	$1{,}9 \cdot 10^{-6}$	43	$1{,}4 \cdot 10^{-8}$	36	$\mathbf{4{,}7 \cdot 10^{-4}}$	65	$\mathbf{7{,}8 \cdot 10^{-6}}$	67
Adagrad	—	—	$5{,}2 \cdot 10^{-2}$	39	$8{,}2 \cdot 10^{-5}$	33	$1{,}1 \cdot 10^{-2}$	63	—	—
Nadam	—	—	$8{,}3 \cdot 10^{-6}$	36	$2{,}1 \cdot 10^{-5}$	36	$1{,}7 \cdot 10^{-3}$	64	$6{,}9 \cdot 10^{-4}$	62
RMSProp	—	—	$6{,}3 \cdot 10^{-5}$	31	$4{,}6 \cdot 10^{-8}$	55	$1{,}1 \cdot 10^{-}$	67	—	—

Нетрудно видеть, что наиболее эффективными являются методы оптимизации типа Adam.

О выборе структуры нейронной сети

Рассмотрим теперь влияние структуры нейронной сети на точность приближенного решения задач (4)–(7) и скорость ее обучения. С этой целью были выбраны три наиболее распространенные архитектуры, схематично изображенные на рис. 4: однослойная модель (а), двухслойная модель с равномерным распределением нейронов в скрытых слоях (б) и трехслойная модель с уменьшающимся количеством нейронов в скрытых слоях (в).

В проведенных экспериментах количество нейронов в скрытом слое однослойной модели равно 40. В этом случае вектор обучаемых параметров $\vec{p}$ содержит 121 элемент. Структуры двухслойной и трехслойной нейронных сетей подбирались так, чтобы количество элементов век-

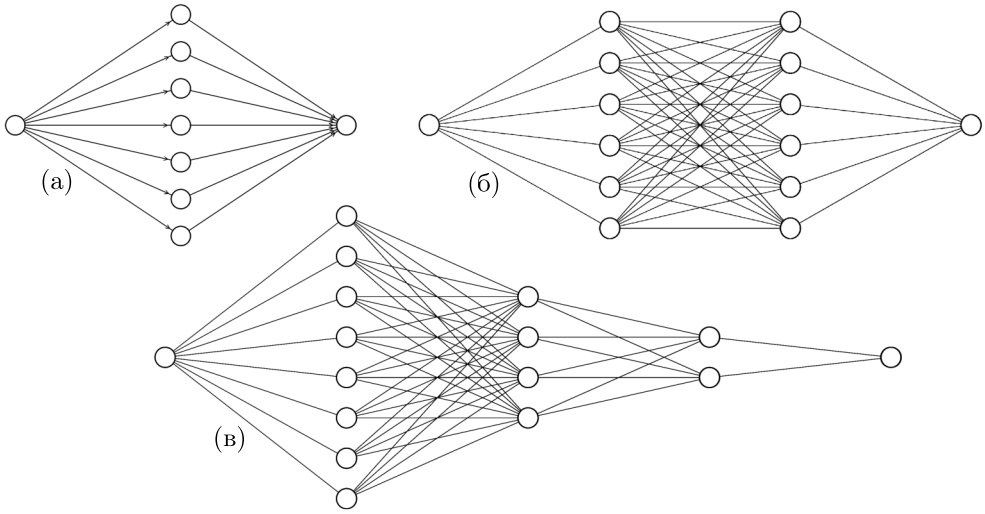

Рис. 4. Архитектуры нейронной сети

тора $\vec{p}$ во всех случаях было примерно одинаково. Так, в двухслойной модели используются два скрытых слоя по 9 нейронов в каждом, так что вектор $\vec{p}$ состоит из 118 параметров. Трехслойная модель содержит 10, 6 и 4 нейронов в скрытых слоях и, соответственно, 119 обучаемых параметров. Каждая модель обучалась с применением двух наиболее эффективных алгоритмов оптимизации — Adam и LBFGS. При использовании первого алгоритма требуется, как и в предыдущих экспериментах, 40 000 эпох. Для обучения методом LBFGS оказывается достаточным 600 эпох, после чего функция потерь попадает в точку локального минимума и алгоритм перестает сходиться.

Результаты расчетов приведены в табл. 5. Как и ранее, прочерки в таблице означают отсутствие сходимости алгоритма и невозможность обучения нейросети.

Таблица 5. Результаты расчетов величин функции потерь L и времени T (с) обучения нейросетей с одним, двумя или тремя скрытыми слоями при решении ОДУ (4)–(7) с использованием алгоритмов оптимизации Adam и LBFGS

Модель	Алгоритм	(4)		(5)		(6)		(7_1)		(7_2)	
		L	T	L	T	L	T	L	T	L	T
1 слой	Adam	$1{,}1 \cdot 10^{-3}$	56	$3{,}2 \cdot 10^{-7}$	40	$9{,}8 \cdot 10^{-9}$	35	$4{,}9 \cdot 10^{-4}$	51	$1{,}3 \cdot 10^{-5}$	60
	LBFGS	—	—	$2{,}1 \cdot 10^{-6}$	4	$5{,}7 \cdot 10^{-7}$	2	$2{,}1 \cdot 10^{-3}$	4	$1{,}1 \cdot 10^{-5}$	7
2 слоя	Adam	—	—	$3{,}1 \cdot 10^{-6}$	45	$1{,}6 \cdot 10^{-8}$	42	$4{,}2 \cdot 10^{-3}$	62	—	—
	LBFGS	—	—	$2{,}3 \cdot 10^{-6}$	5	$8{,}7 \cdot 10^{-7}$	2	$9{,}9 \cdot 10^{-4}$	10	—	—
3 слоя	Adam	—	—	$8{,}2 \cdot 10^{-1}$	52	$9{,}9 \cdot 10^{-9}$	50	$5{,}7 \cdot 10^{-3}$	73	—	—
	LBFGS	—	—	$9{,}1 \cdot 10^{-5}$	4	$1{,}1 \cdot 10^{-6}$	3	$8{,}9 \cdot 10^{-3}$	6	—	—

Анализ вычислительных экспериментов показывает, что алгоритм LBFGS требует на порядок меньшего времени обучения T, чем метод Adam, но по значению L функции потерь может на порядки уступать ему. Как отмечалось во введении, наилучшие результаты достигаются в случаях обучения двух- и трехслойных нейронных сетей с помощью алгоритма LBFGS. Однако, как показывают приведенные в таблице данные, усложнение структуры нейронный сети не обеспечивает повышения точности решения подобных типов задач.

Это обусловлено тем, что при решении задачи оптимизации градиенты, вычисляемые от входного слоя к входному и используемые для пересчета весов, могут становиться очень ма-

ленькими или очень большими. В первом случае веса обновляются незначительно, что приводит к очень медленной сходимости процесса оптимизация. Во втором случае большие значения градиентов могут вызвать нестабильность обучения и отсутствие сходимости алгоритма. Все это значительно усложняет процесс обучения многослойных нейронных сетей и требует тщательной настройки гиперпараметров модели и самих алгоритмов оптимизации. Использование же однослойной нейронной сети с применением метода Adam позволяет с большей точностью построить нейросетевое решение во всех рассмотренных случаях, хотя это может потребовать больших временных затрат на обучение модели.

О построении функции нейросетевого решения и функции потерь

Перейдем к изучению способов построения приближенных нейросетевых решений. В одном из подходов нейросетевое решение $\widetilde{y}(t)$ будем искать по формулам [Lee, Kang, 1990]

$$\widetilde{y}(t) = \Upsilon(t, \vec{p}), \quad L(\vec{\tau}, \vec{p}) = \frac{1}{2}\left(\frac{1}{N_D - 2}\sum_{i=1}^{N_D}\left(\frac{d\widetilde{y}(t_i)}{dt} - f(t_i, \widetilde{y}(t_i))\right)^2 + \left(\widetilde{y}(t_0) - y_0\right)^2\right), \tag{8}$$

где t_i — компоненты вектора $\vec{\tau}$ набора обучения $\vec{\tau} = \left(t_0, t_1, \ldots, t_{N_D}\right)$, $t_{N_D} = t_N$. В общем случае точки t_i вектора $\vec{\tau}$ отличаются от точек дискретной области численного решения задачи Коши, количество N которых $N \gg N_D$.

В другом подходе будем использовать формулы [Flamant, Pavlos, Sondak, 2020]

$$\widetilde{\widetilde{y}}(t, y_0) = y_0 + \alpha(t)\Upsilon(t, \vec{p}), \quad L(\vec{\tau}, \vec{p}) = \frac{1}{N_D - 1}\sum_{i=0}^{N_D}\alpha(t_i)\left(\frac{d\widetilde{\widetilde{y}}(t_i)}{dt} - f(t_i, \widetilde{\widetilde{y}}(t_i))\right)^2. \tag{9}$$

Здесь $\alpha(t) = \frac{t - t_0}{t_N - t_0}$ — переменный весовой коэффициент, $0 \leqslant \alpha(t) \leqslant 1$.

Формулы (8) и (9) отличаются как по способу использования выходного значения нейросети $\Upsilon(t, \vec{p})$, так и по виду функции потерь $L(\vec{\tau}, \vec{p})$. В первом случае граничное значение y_0 включено в функцию потерь L, а во втором оно входит в определение приближенного нейросетевого решения $\widetilde{\widetilde{y}}$. Второй подход по сравнению с первым обладает преимуществом при обучении нейросети в тех точках t_i интервала $[t_0, t_N]$, которые расположены вблизи конечной точки t_N. Это особенно необходимо при решении задач, в которых функция $y(t)$ является осциллирующей и монотонно убывающей. Именно таким является точное решение задачи (7_1), фрагмент которого вблизи точки $t_N = 6\pi$ показан точками на рис. 5. Поэтому именно эта задача была выбрана в качестве примера для оценки возможностей каждого из подходов.

Вычислительные эксперименты проводились при тех же исходных данных, а варьировался лишь способ построения нейросетевого решения. На рис. 5, *а* пунктирной линией показано решение $\widetilde{y}$, а сплошной — $\widetilde{\widetilde{y}}$. Нетрудно видеть, что во втором случае нейронная сеть хорошо учитывает особенности правой части уравнения (7_1), которая представляет собой периодическую функцию с затухающей амплитудой колебаний, и приближенное решение качественно и количественно повторяет точное решение. В первом же случае по мере уменьшения значений осциллирующей функции $y(t)$ нейросетевое решение $\widetilde{y}$ перестает сходиться к точному решению.

Рисунок 5, *б* демонстрирует различия погрешностей методов. Линиями 1 и 2 показаны погрешности расчетов $\widetilde{\widetilde{e}}(t) = |y(t) - \widetilde{\widetilde{y}}(t)|$ и $\widetilde{e}(t) = |y(t) - \widetilde{y}(t)|$ по формулам (9) и (8) соответственно при $N_D = 300$, пунктирной линией 3 — погрешность $\overline{e}(t) = |y(t) - \overline{y}(t)|$ метода Эйлера при $N = 2500$. Нетрудно видеть, что второй подход по сравнению с первым обеспечивает наиболее высокую точность построения приближенного решения при практически одинаковом времени обучения.

Отметим, что результаты аналогичных исследований решений задач (4), (5), (6) и (7_2) показывают, что для них эффективность обоих подходов одинакова.

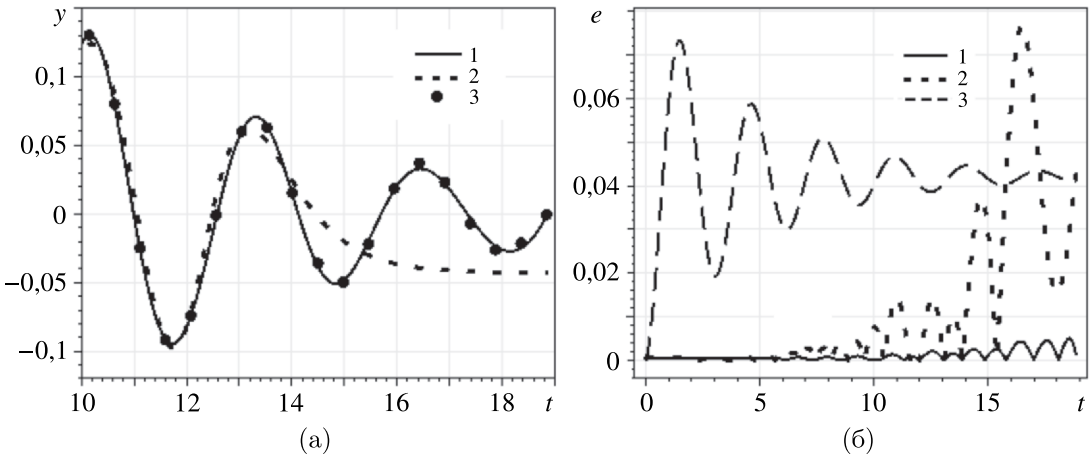

Рис. 5. а) Графики нейросетевых $\widetilde{\widetilde{y}}$ (1), $\widetilde{y}$ (2) и точного y (3) решений задачи (7_1); б) графики погрешности нейросетевых $\widetilde{\widetilde{e}}$ (1), $\widetilde{e}$ (2) решений и численного (3) решения по методу Эйлера

Выводы

Исследованы методы построения приближенных нейросетевых решений обыкновенных дифференциальных уравнений на примере задач Коши с различным типом правой части. На основе вычислительных экспериментов показано:

- применение языка программирования C# и библиотеки TorchSharp при разработке программных модулей, реализующих физически информированные нейронные сети, позволяет сократить время их обучения на 20–40 % по сравнению с аналогичными средствами языка Python и библиотек TensorFlow и PyTorch;
- наиболее эффективными для построения нейросетевых решений в рассматриваемых задачах являются функции активации сигмоида и гиперболический тангенс;
- существует такое количество нейронов скрытого слоя однослойной нейронной сети, при котором достигается наименьшее значение функции потерь за минимальное время обучения нейросетевой модели, причем дальнейшее усложнение структуры нейронной сети за счет увеличения числа нейронов не приводит к существенному улучшению результатов обучения;
- величина шага сетки между точками обучающей выборки для всех этих оптимальных случаев (при достижении минимума функцией потерь) практически одинакова и равна ~ 0,05;
- для обучения однослойных нейронных сетей при решении задач оптимизации наиболее эффективными являются метод Adam и его модификации;
- для обучения двух- и трехслойных нейронных сетей, как и ожидалось, использование алгоритма LBFGS в ряде случаев требует на порядок меньшего времени обучения при достижении одинакового порядка точности, чем применение метода Adam;
- построение нейросетевого решения с применением переменных весовых коэффициентов в задачах Коши с осциллирующим и монотонно убывающим решением обладает преимуществом при обучении нейросети в тех узлах, которые расположены вблизи конечной точки интервала решения задачи.

Список литературы (References)

Бурнаев Е. В., Бернштейн А. В., Вановский В. В., Зайцев А. А., Булкин А. М., Игнатьев В. Ю., Шадрин Д. Г., Илларионова С. В., Оселедец И. В., Михалев А. Ю., Осипцов А. А., Артемов А. А., Шараев М. Г., Трофимов И. Е. Фундаментальные исследования и разработки в области прикладного искусственного интеллекта // Доклады Российской академии наук. Математика, информатика, процессы управления. — 2022. — Т. 508, № 1. — С. 19–27.
Burnaev E. V., Bernstein A. V., Vanovsky V. V., Zaitsev A. A., Bulkin A. M., Ignatiev V. Yu., Shadrin D. G., Illarionova S. V., Oseledets I. V., Mikhalyov A. Yu., Osiptsov A. A., Artemov A. A., Sharaev M. G., Trofimov I. E. Fundamental research and development in the field of applied artificial intelligence // Doklady Mathematics. — 2023. — Vol. 106, Issue 1. — P. 14–22. (Original Russian paper: *Burnaev E. V., Bernstein A. V., Vanovsky V. V., Zaitsev A. A., Bulkin A. M., Ignatiev V. Yu., Shadrin D. G., Illarionova S. V., Oseledets I. V., Mikhalyov A. Yu., Osiptsov A. A., Artemov A. A., Sharaev M. G., Trofimov I. E.* Fundamental'nye issledovania i razrabotki v oblasti prikladnogo iskusstvennogo intellekta // Doklady Rossijskoj akademii nauk. Matematika, informatika, processy upravlenia. — 2022. — Vol. 508, No. 1. — P. 19–27.)

Конюхов В. М. Дисперсные потоки в нефтяных скважинах. — Казань: Изд-во КГУ, 1990. — 140 с.
Konyukhov V. M. Dispersnye potoki v neftyanykh skvazhinah [Dispersed flows in oil wells]. — Kazan': Izd-vo KGU, 1990. — 140 p. (in Russian).

Семенов И. А. Использование искусственных нейронных сетей для поиска решений дифференциальных уравнений // Сборник научных трудов Ангарского государственного технического университета. — 2024. — № 21. — С. 76–79.
Semenov I. A. Ispol'zovanie iskusstvennykh nejronnykh setej dlya poiska reshenij differencial'nykh uravnenij [Using artificial neural networks to find the solutions of differential equations] // Sbornik nauchnykh trudov Angarskogo gosudarstvennogo tehnicheskogo universiteta. — 2024. — No. 21. — P. 76–79 (in Russian).

Тюрин К. А. Решение дифференциального уравнения первого порядка с помощью нейронной сети // Процессы управления и устойчивость. — 2019. — Т. 6, № 1. — С. 373–377.
Turin K. A. Reshenie differencial'nogo uravnenia pervogo poryadka s pomoschiu neironnoj seti [Solving a first order differential equation using a neural network] // Processy upravlenia i ustojchivost. — 2019. — Vol. 6, No. 1. — P. 373–377 (in Russian).

Baydin A. G., Pearlmutter B. A., Radul A. A., Siskind J. M. Automatic differentiation in machine learning: a survey // Journal of Machine Learning Research. — 2018. — Vol. 18, No. 153. — P. 1–43.

Dami C., Shallue C. J., Zachary N., Jaehoon L., Maddison C. J., Dahl G. E. On empirical comparisons of optimizers for deep learning // arXiv preprint. — 2019. — arXiv:1910.05446

Flamant C., Pavlos P., Sondak D. Solving differential equations using neural network solution bundles // arXiv preprint. — 2020. — arXiv:2006.14372

Gorikhovskii V. I., Evdokimova T. O., Poletansky V. A. Neural networks in solving differential equations // Journal of Physics: Conference Series. — 2022. — P. 1–10.

Hornik K., Stinchcombe M., White H. Multilayer feedforward networks are universal approximators // Neural Networks. — 1989. — Vol. 2, No. 5. — P. 359–366.

Lagaris I. E., Likas A., Fotiadis D. I. Artificial neural networks for solving ordinary and partial differential equations // IEEE transactions on neural networks. — 1998. — Vol. 5, No. 9. — P. 987–1000.

Lee H., Kang I. S. Neural algorithm for solving differential equations // Journal of Computational Physics. — 1990. — No. 91. — P. 110–131.

Liu D. C., Nocedal J. On the limited memory BFGS method for large scale optimization // Mathematical programming. — 1989. — Vol. 45, No. 1. — P. 503–528.

Paszke A., Gross S., Sam C., Chintala S., Chanan G., Yong E., DeVito Z., Lin Z., Desmaison A., Antiga L., Lerer A. Automatic differentiation in PyTorch // NIPS 2017 Workshop Proceedings. — 2019. — P. 1–4.

Petriu E. Neural networks: basics. — School of Electrical Engineering and Computer Science, University of Ottawa, Canada, 2004. — 42 p.

Sagar S., Simone S., Anidhya A. Activation functions in neural networks // Towards Data Sci. — 2017. — Vol. 10, No. 12. — P. 310–316.

Shiruru K. An introduction to artificial neural network // International Journal of Advance Research and Innovative Ideas in Education. — 2016. — Vol. 1, No. 5. — P. 27–30.

Süli E., Mayers D. An introduction to numerical analysis. — Cambridge: Cambridge university press, 2003. — 429 p.