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

Диков Александр Евгеньевич
Б20-221

Научный руководитель:
Д. П. Макаревич, ООО "РЦР"

Содержание

Введение

Наиболее актуальные статьи по решению дифференциальных уравнений с помощью нейронных сетей можно разделить на несколько категорий.

- К первой можно отнести те статьи, в которых описаны основные принципы PINN, это понятие вводится.
- Ко второй категории отнесем стати, описывающие сопутствующие технологии, такие как, например, автоматическое дифференцирование.
- Третья категория статей освещает вопросы сходимости, вводя необходимые теоремы, доказывающие непротиворечивость PINN.
- В четвертой категории статьи, описывающие различные приемы и идеи, по ускорению сходимости или улучшению качества решения. Обычно они касаются изменения функции активации, общей архитектуры или функции потерь.

Physics-informed neural networks

В этой статье впервые был представлен термин Physics-informed neural networks (PINN)*. Заложены многие принципы и решения, которые впоследствии были использованы и другими авторами. Пусть есть задача:

$$u_t + \mathcal{N}[u] = 0, x \in \omega, t \in [0, T], \tag{1}$$

Введем f как левую часть уравнения:

$$f := u_t + \mathcal{N}[u] \tag{2}$$

Введем компоненты функции потерь:

$$MSE = MSE_u + MSE_f,$$

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u(t_u^i, x_u^i) - u^i \right|^2, \tag{3}$$

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f(t_f^i, x_f^i) \right|^2.$$

Physics-informed neural networks

Авторами было проведено несколько исследований на различных моделях и различных уравнениях. Во всех случая метод проявил себя хорошо. Отдельного внимания заслуживает таблица сравнения архитектур - подобное исследование, для более современной архитектуры и уравнения теплопроводности я приводил ранее.

Table A.2
Burgers' equation: Relative $\mathbb{L}_2$ error between the predicted and the exact solution $u(t, x)$ for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to $N_u = 100$ and $N_f = 10{,}000$, respectively.

Layers \ Neurons	10	20	40
2	7.4e−02	5.3e−02	1.0e−01
4	3.0e−03	9.4e−04	6.4e−04
6	9.6e−03	1.3e−03	6.1e−04
8	2.5e−03	9.6e−04	5.6e−04

Рис. 1: Зависимость L2 меры от архитектуры

Table A.1
Burgers' equation: Relative $\mathbb{L}_2$ error between the predicted and the exact solution $u(t, x)$ for different number of initial and boundary training data N_u, and different number of collocation points N_f. Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e−01	4.4e−01	8.9e−01	1.2e+00	9.9e−02	4.2e−02
40	6.5e−02	1.1e−02	5.0e−01	9.6e−03	4.6e−01	7.5e−02
60	3.6e−01	1.2e−02	1.7e−01	5.9e−03	1.9e−03	8.2e−03
80	5.5e−03	1.0e−03	3.2e−03	7.8e−03	4.9e−02	4.5e−03
100	6.6e−02	2.7e−01	7.2e−03	6.8e−04	2.2e−03	6.7e−04
200	1.5e−01	2.3e−03	8.2e−04	8.9e−04	6.1e−04	4.9e−04

Рис. 2: Зависимость L2 меры от числа точек

*M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019.

Automatic Differentiation in Machine Learning

В статье "Automatic Differentiation in Machine Learning: a Survey"рассматривается техника автоматического дифференцирования (AD). AD относится к семейству методов, которые вычисляют производные путем накопления значений во время выполнения кода для создания числовых оценок производных, а не с помощью конкретных выражений.
В то время как численное дифференцирование может быть неустойчиво, использовать подход символьного дифференцирования для некоторой сложной функции f(x) будет вызывать возникновение вложенных дубликаты любых вычислений, такой подход может легко привести к появлению экспоненциально больших выражений.

Automatic Differentiation in Machine Learning

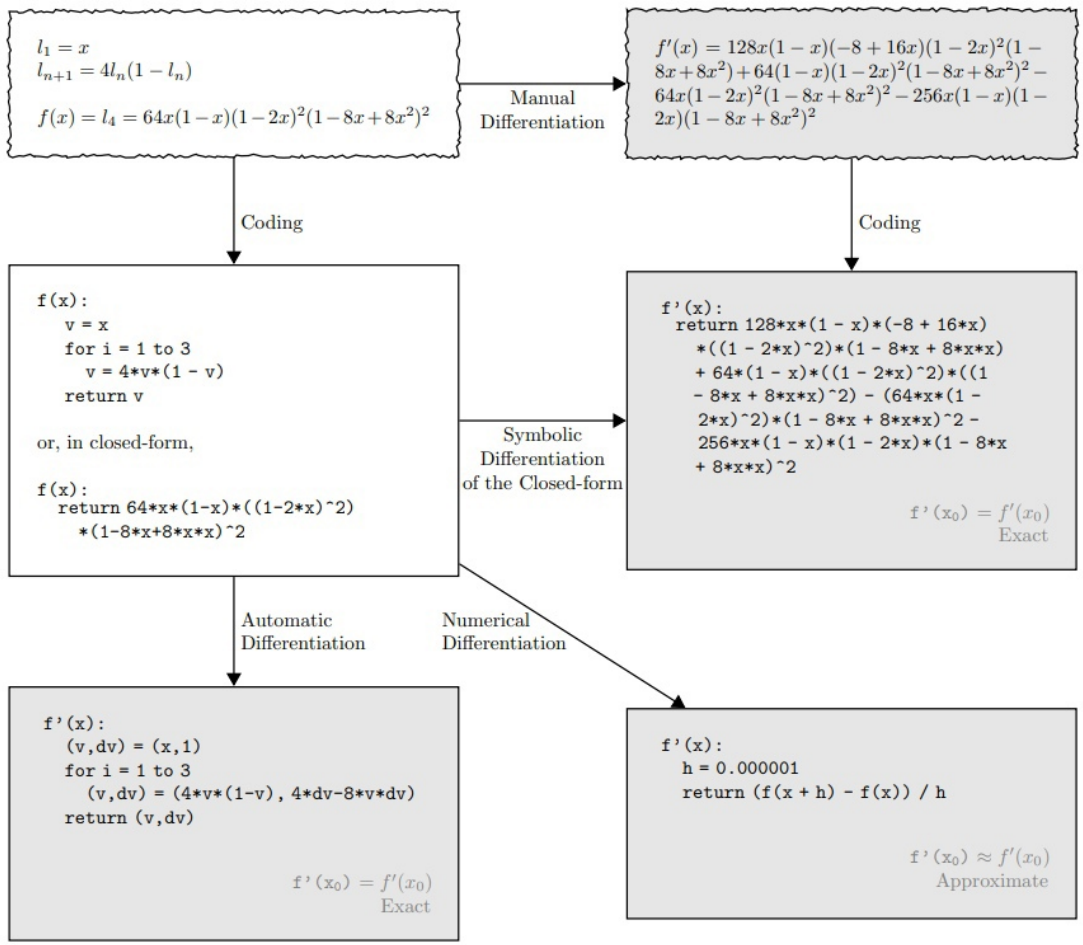

Рис. 3: Отличие автоматического дифференцирования от численного и символьного

В основе AD лежит следующий подход: применять символьное дифференцирование на уровне элементарных операций и сохранять промежуточные численные результаты.

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

Таблица 1: Пример вычислений

DeepXDE: A Deep Learning Library for Solving Differential Equations

В этой статье метод был описан еще более подробно и предложено сразу множество идей по улучшению:

- Можно добавлять настраиваемые веса к потерям с разных точек.
- Необходим подбор параметров архитектуры сети, для этого можно использовать callback функцию.
- Точки для вычисления остатков можно выбирать разными способами:
  - выбрать их один раз в начале (случайно или сеткой) и не менять.
  - выбирать на каждой итерации обучения разные точки.
  - выбирать точки, улучшая их расположение в зависимости от величины функции потерь.

Algorithm 1 Улучшение распределения остаточных точек для обучения (RAR)

1: Выберите начальные точки и обучите нейронную сеть ограниченное количество итераций.
2: Оцените средний остаток ДУ $\mathcal{E}_r$ с использованием метода Монте-Карло. Это делается усреднением значений по набору случайно выбранных наборов точек $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{|\mathcal{S}|}\}$:

$$\mathcal{E}_r \approx \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) \right\|. \quad (4)$$

3: Если $\mathcal{E}_r < \mathcal{E}_0$, завершите процедуру. В противном случае добавьте новые точки m с наибольшими остатками из $\mathcal{S}$ к точкам $\mathcal{T}$, повторно обучите сеть и вернитесь к Шагу 2.

DeepXDE: A Deep Learning Library for Solving Differential Equations

Кроме того, в статье освещается вопрос ошибок и эта терминалогия будет использована и в дальнейших работах. Ошибки при решении связаны с различными факторами:

- Аппроксимация - разница между действительным решением и самой ближайшей из функций, которую можно выбрать из всего семейста функций, которые могут быть представлены выбранной сетью.
- Оценка - связана с конечностью набора точек для обучения.
- Генерализация - связанная с выбором точек (их положением и количеством) в которых находятся остатки. Ошибки аппроксимации и оценки вместе дают ошибку генерализации.
- Оптимизатия - связана с тем, что оптимизатор ищет лучшую функцию из семейства, но находит лишь ее аппроксимацию, с точностью зависящей от learning rate и числа шагов.

Более сложная сеть будет обладать меньшей ошибкой аппроксимации и большей ошибкой генерализации - это явление изместно и в математической статистике как bias-variance tradeoff. Над данный момент ошибки - это лишь теоретические оценки, вычислить их значения невозможно на данный момент.

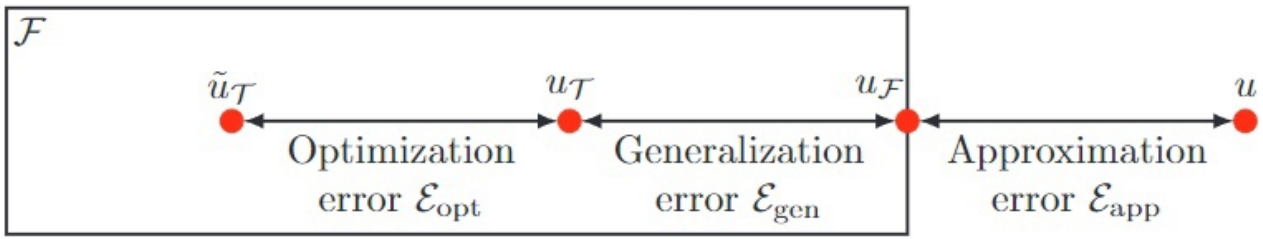

Рис. 4: Виды ошибок

Implicit Neural Representations with Periodic Activation Functions

Рассмотрим функции класса:

$$F\left(\mathbf{x}, \Phi, \nabla_{\mathbf{x}}\Phi, \nabla^2_{\mathbf{x}}\Phi, \ldots\right) = 0, \quad \Phi : \mathbf{x} \mapsto \Phi(\mathbf{x}). \tag{5}$$

Будем искать решение общей задачи среди функций этого класса. Представляем это как задачу существования, где ищется функция Φ, которая удовлетворяет набору ограничений M включающем $\{C_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla\Phi(\mathbf{x}), \ldots)\}_{m=1}^{M}$, каждому из которых соответствует функция Φ и/или его производные к величинам $\mathbf{a}(\mathbf{x})$:

$$\text{Ищем } \Phi(\mathbf{x}), \text{ при этом } C_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla\Phi(\mathbf{x}), \ldots) = 0, \forall \mathbf{x} \in \Omega_m, m = 1, \ldots \tag{6}$$

Эту проблему можно выразить с помощью функции потерь, которая штрафует отклонения от каждого из ограничений в своей области Ω_m :

$$\mathcal{L} = \int_{\Omega} \sum_{m=1}^{M} \mathbf{1}_{\Omega_m}(\mathbf{x}) \left\| C_m(\mathbf{a}(\mathbf{x}), \Phi(\mathbf{x}), \nabla\Phi(\mathbf{x}), \ldots) \right\| d\mathbf{x}, \tag{7}$$

с индикаторной функцией $1_{\Omega_m}(\mathbf{x}) = 1$ с $\mathbf{x} \in \Omega_m$ и 0 когда $\mathbf{x} \notin \Omega_m$. На практике функция потерь реализуется путем выборки Ω. Набор данных $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{a}_i(\mathbf{x}))\}_i$ представляет собой набор кортежей координат $\mathbf{x}_i \in \Omega$ вместе с выборками величин $\mathbf{a}(\mathbf{x}_i)$ которые фигурируют в ограничениях. Таким образом, потери в уравнении применяются к координатам $\mathbf{x}_i$ выбранным из набора данных, что дает потери $\tilde{\mathcal{L}} = \sum_{i \in \mathcal{D}} \sum_{m=1}^{M} \left\| C_m\left(a\left(\mathbf{x}_i\right), \Phi\left(\mathbf{x}_i\right), \nabla\Phi\left(\mathbf{x}_i\right), \ldots\right) \right\|$. На

практике набор данных $\mathcal{D}$ отбирается динамически во время обучения, аппроксимируя $\mathcal{L}$ лучше по мере роста количества выборок, как при интегрировании Монте-Карло.

$$\Phi(\mathbf{x}) = \mathbf{W}_n \left(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0\right)(\mathbf{x}) + \mathbf{b}_n, \quad \mathbf{x}_i \mapsto \phi_i\left(\mathbf{x}_i\right) = \sin\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right). \tag{8}$$

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

В статье рассматриваются параметрические функции активации, названные адаптивными.

- Функции активации активирует конкретный нейрон.
- Без функции активации веса и смещение выполняют простое линейное преобразование, что соответствует случаю линейной регрессионной модели.
- Нелинейная функция активации позволяет модели решать более сложные задачи.
- Функции активации делают алгоритм обратного распространения ошибки возможным.
- Необходимо выбирать функцию активации, менее подверженную проблеме затухающего и взрывающегося градиента.

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

$$\text{Sigmoid:} \quad \frac{1}{1+e^{-ax}}, \quad \text{Hyperbolic tangent:} \quad \frac{e^{ax}-e^{-ax}}{e^{ax}+e^{-ax}},$$
$$\text{ReLU:} \quad \max(0, ax), \quad \text{Leaky ReLU:} \quad \max(0, ax) - v\max(0, -ax). \tag{9}$$

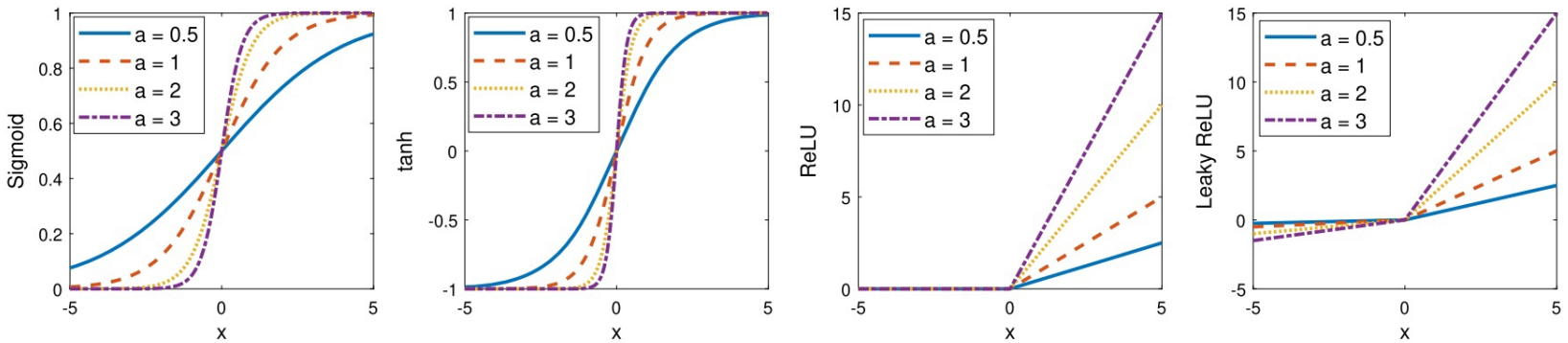

Рис. 5: Графики адаптивных функций активации

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

Пример изменения функции в процессе обучения в случае функции активации *tanh*.

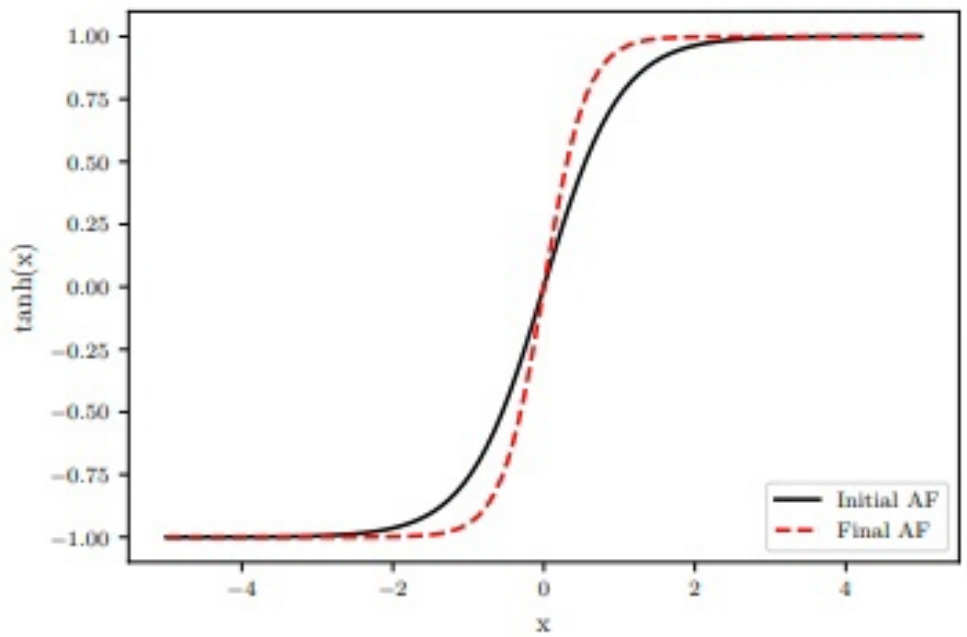

Рис. 6: Начальная и конечная функция

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

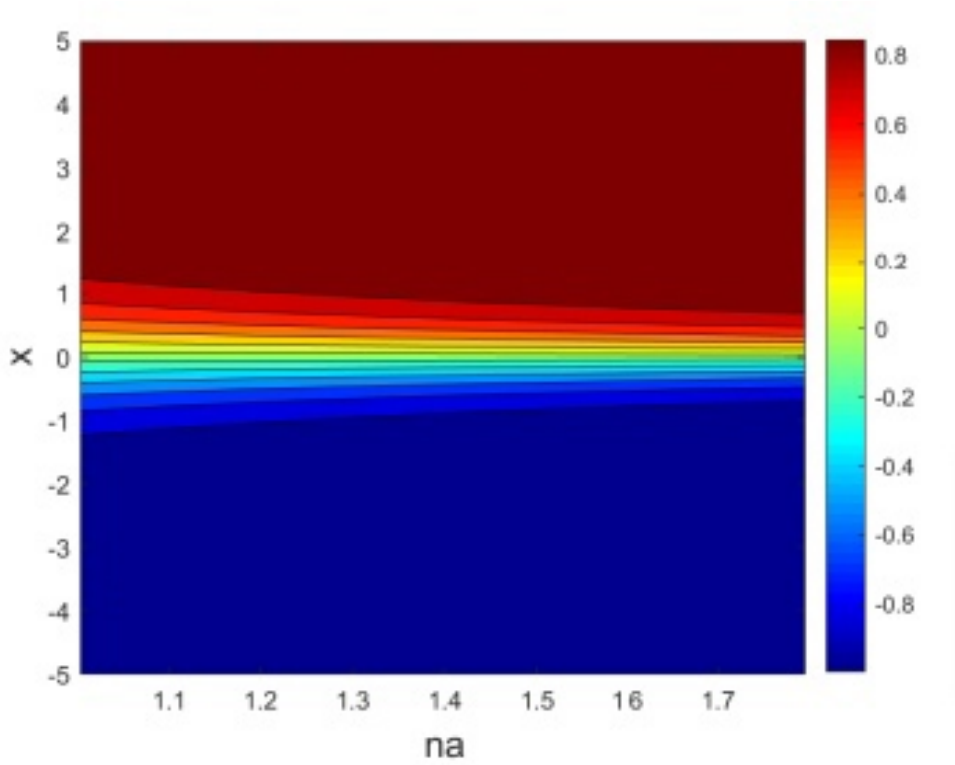

Рис. 7: Плоскость активации

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

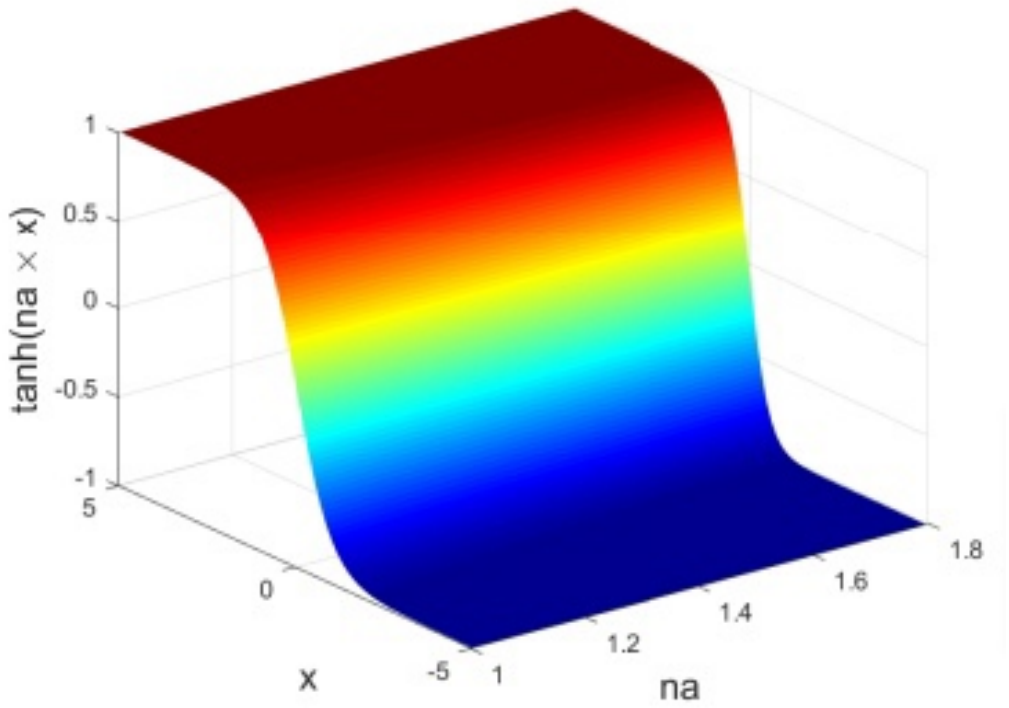

Рис. 8: Поверхность активации

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

Пример изменения функции в процессе обучения в случае функции активации *sin*.

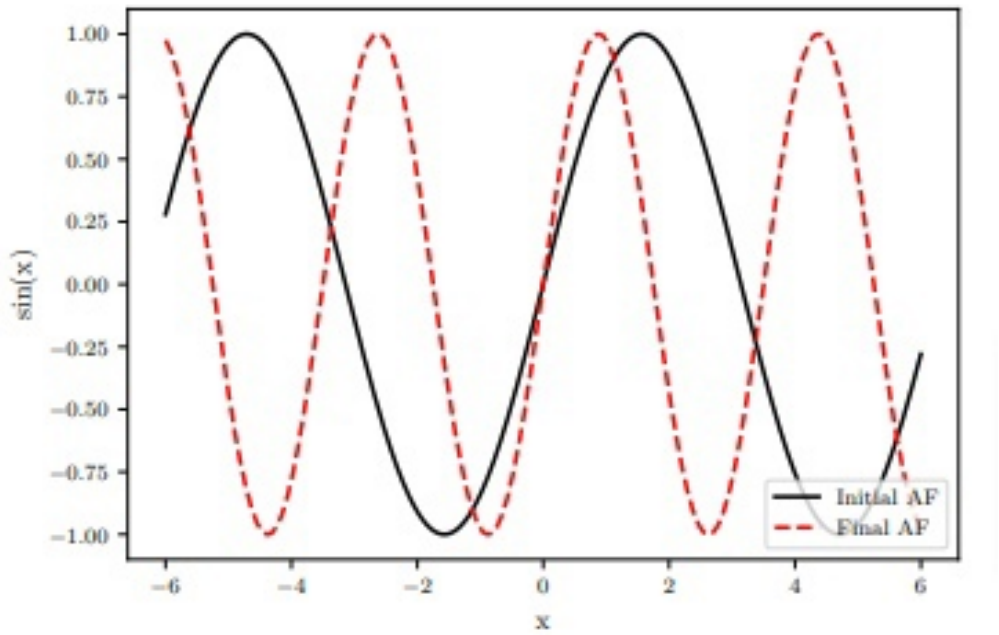

Рис. 9: Начальная и конечная функция

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

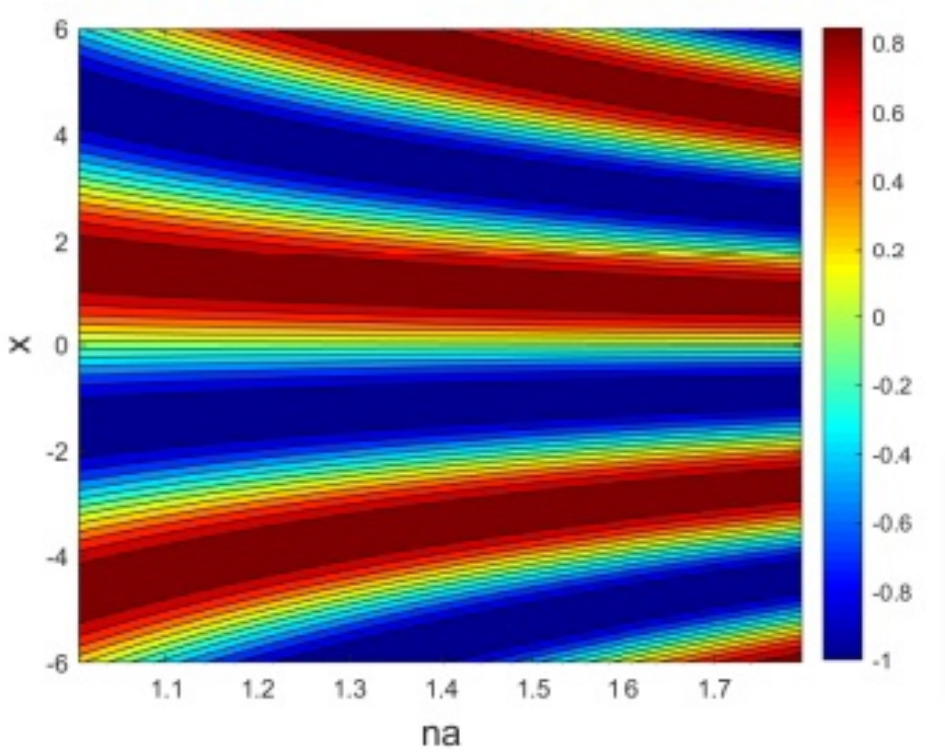

Рис. 10: Плоскость активации

Adaptive activation functions accelerate convergence in deep and physics-informed neural networks

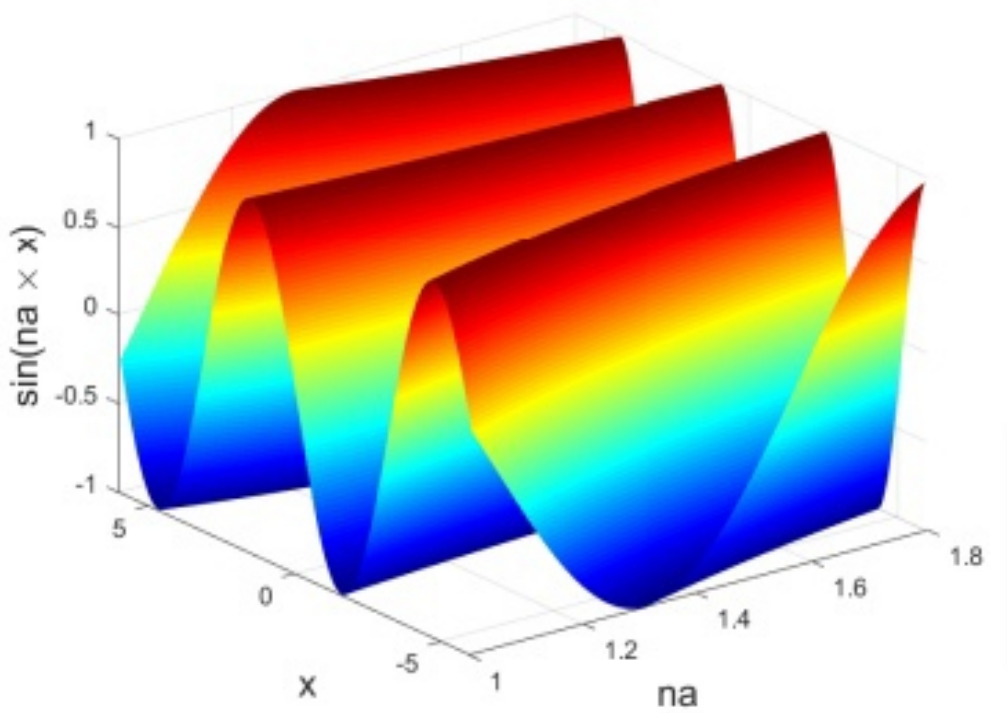

Рис. 11: Поверхность активации

Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks

В статье рассматривается два подхода к локально-адаптивным функциям активации:

- Послойные - L-LAAF.
  $\sigma\left(na^k \mathcal{L}_k\left(z^{k-1}\right)\right), \quad k = 1, 2, \cdots, D-1$
- Понейронные - N-LAAF.
  $\sigma\left(na_i^k \left(\mathcal{L}_k\left(z^{k-1}\right)\right)_i\right), \quad k = 1, 2, \cdots, D-1, \quad i = 1, 2, \cdots, N_k$

Чтобы получить ускорение в сходимости метода, в функцию потерь добавляется член восстановления наклона на основе наклона функции активации:

$$S(a) = \begin{cases} \dfrac{1}{\frac{1}{D-1}\sum_{k=1}^{D-1} exp(a^k)} & \text{для L-LAAF} \\ \dfrac{1}{\frac{1}{D-1}\sum_{k=1}^{D-1} exp(\frac{\sum_{i=1}^{N_k} a_i^k}{N_k})} & \text{для N-LAAF} \end{cases}$$

Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks

Ряд математических следствий из статьи:

- Доказывается, что алгоритмы градиентного спуска не стягиваются к субоптимальным критическим точкам или локальным минимумам при практических условиях.
- Доказывается, что динамика градиента предложенного метода недостижима базовыми методами с любыми (адаптивными) темпами обучения.
- Демонстрируется, что адаптивные методы ускоряют сходимость, неявно умножая матрицы условий на градиент базового метода без явного вычисления матрицы условий.
- Показано, что различные адаптивные функции активации порождают различные неявные матрицы условий, а предложенные методы с восстановлением наклона ускоряют процесс обучения.

Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks

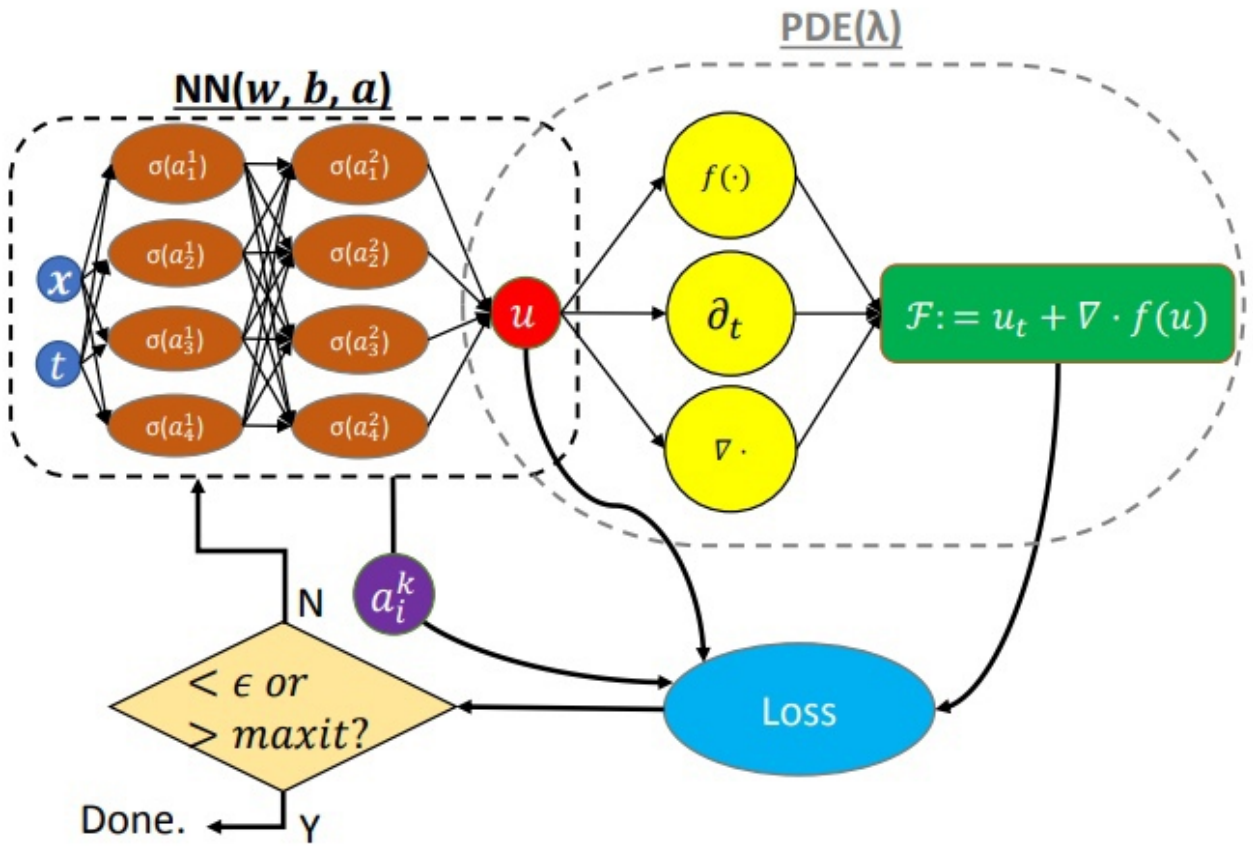

Рис. 12: Схема сети с LAAF для уравнения Бюргерса

Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks

Algorithm 2 LAAF-PINN алгоритм

1: Обучающие данные: $u_{\hat{\Theta}}$ для $\left\{\mathbf{x}_u^i\right\}_{i=1}^{N_u}$, Остатки: $\mathcal{F}_{\hat{\mathbf{\Theta}}}$ для $\left\{\mathbf{x}_f^i\right\}_{i=1}^{N_f}$

2: Создайте нейронную сеть $u_{\hat{\Theta}}$ с параметрами $\hat{\mathbf{\Theta}}$.

3: Создайте остаточную нейронную есть $\mathcal{F}_{\hat{\mathbf{\Theta}}}$ путем подстановки $u_{\hat{\Theta}}$ в основные уравнения.

4: Введите функцию потерь:

$$\tilde{J}(\hat{\mathbf{\Theta}}) = \frac{W_{\mathcal{F}}}{N_f}\sum_{i=1}^{N_f}\left|\mathcal{F}_{\hat{\mathbf{\Theta}}}\left(\mathbf{x}_f^i\right)\right|^2 + \frac{W_u}{N_u}\sum_{i=1}^{N_u}\left|u^i - u_{\hat{\mathbf{\Theta}}}\left(\mathbf{x}_u^i\right)\right|^2 + W_a\mathcal{S}(a),$$

5: Найдите лучшие параметры $\hat{\mathbf{\Theta}}^*$ используя подходящий метод оптимизации для минимизации функции потерь $\tilde{J}(\hat{\mathbf{\Theta}})$ как

$$\hat{\mathbf{\Theta}}^* = \underset{\hat{\mathbf{\Theta}}\in\hat{\mathcal{V}}}{\arg\min}\tilde{J}(\hat{\mathbf{\Theta}})$$

On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs

Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs

Спасибо за внимание