Использование нейронных сетей для нахождения решения уравнения Бюргерса

Коновалов Артём 304

17 мая 2021 г.

0.1 Проблематика

Дифференцальные уравнения в частных производных используются во многих областях нашей жизни. С их помощью описываются многие физические процессы, также они активно применяются в инженерии и финансах. Численное решение уравнений в частных производных сводится к построению сетки и замене производных разностными схемами. Подобный способ становится вычислительно затратным в высоких измерениях из-за резкого увеличения количества точек сетки.
Последние достижения в областях искусственного интеллекта и машинного обучения нашли свое применение во многих научных дисциплинах, поэтому, возможно, стоит исследовать полезность алгоритмов глубокого обучения при решении дифференциальных уравнений в частных производных.

0.2 Постановка задачи

Дано уравнение в частных производных вида:

$$\begin{cases} u_t(t,x) + L(u(t,x)) = 0, & (t,x) \in [0,T] \times \Omega \\ u(0,x) = u_0(x), & x \in \Omega \\ u(t,x) = g(t,x), & (t,x) \in [0,T] \times \partial\Omega \end{cases} \tag{1}$$

Требуется реализовать алгоритм, который будет аппроксимировать решение $u(t,x)$ уравнения (1) нейронной сетью $\bar{u}(t,x,\theta)$, где $\theta \in R^d$ являются параметнрами сети.
Далее, следует построить решение, полученное в результате работы нейронной сети и сравнить его с численным решением, полученным методом разностных схем (либо аналитическим решением)

0.3 Общая реализация

Задачей является найти такие параметры θ, чтобы разница $u(t,x) - \bar{u}(t,x,\theta)$ была минимальной. Добиться этого результата можно минимизируя функцию ошибки(функцию потерь) следующего вида:

$$\Im(\bar{u}) = \|\bar{u}_t + L\bar{u}\|^2_{[0,T]\times\partial\Omega} + \|\bar{u} - g\|^2_{[0,T]\times\partial\Omega} + \|\bar{u} - u_0\|^2_{\Omega}$$

Функция $\Im(\bar{u})$ измеряет насколько хорошо аппроксимация решения $\bar{u}$ удовлетворяет дифференциальному оператору, начальным условиям и

граничным условиям.
Сама минимизация функции ошибки происходит при помощи градиентного спуска. Общий алгоритм выглядит следующим образом:

1. Генирируются случайные точки (t_n, x_n) из области определения переменных t и x. Затем генерируются точки (τ_n, z_n) из области определения t и границы области определения x. Также генирируются точки ξ из области определения x. $s_n = (t_n, x_n), (\tau_n, z_n), \xi_n$

2. Считается Функция ошибки:
$G(\theta_n, s_n) = (f(t_n, x_n; \theta_n))^2 + (u(\tau_n, z_n; \theta_n) - g(\tau_n, z_n))^2 + (f(0, \xi; \theta_n) - u_0(\xi_n))^2$

3. Делается шаг градиентного метода:
$\theta_{n+1} = \theta_n - \alpha \bigtriangledown_\theta G(\theta_n, s_n)$

4. Повторять до нужной точности

0.4 Реализация

В качестве примера для исследования было выбрано одномерное уравнение Бюргерса, которое возникает в различных областях прикладной математики(газовая динамика, нелинейная акустика, механика жидкости):

$$\begin{cases} u_t + uu_x = \frac{0.01}{\pi} u_{xx} \\ u(0,x) = -sin(\pi x) \qquad x \in [-1,1]; t \in [0,10] \\ u(t,-1) = u(t,1) = 0 \end{cases}$$

Алгоритм должен аппроксимировать решение $u(t,x)$ нейронной сетью $\bar{u}(t,x,\theta)$, где $\theta \in R^d$ являются параметнрами сети.

$\bar{u}(t,x,\theta) = NeuralNetwork(t, x, weights, biases)$
$u = \bar{u}$
$f(t,x) = u_t + uu_x - \frac{0.01}{\pi} u_{xx}$

Функция потерь для уравнения Бюргерса:

$$J = \|f(t,x)\|^2 + \|\bar{u}(t,-1,\theta)\|^2 + \|\bar{u}(t,1,\theta)\|^2 + \|\bar{u}(t,x,\theta) + sin(\pi x)\|^2$$

С помощью алгоритма грдиентного спуска ищем параметры θ минимизирующие функцию потерь J.

0.5 Инструменты

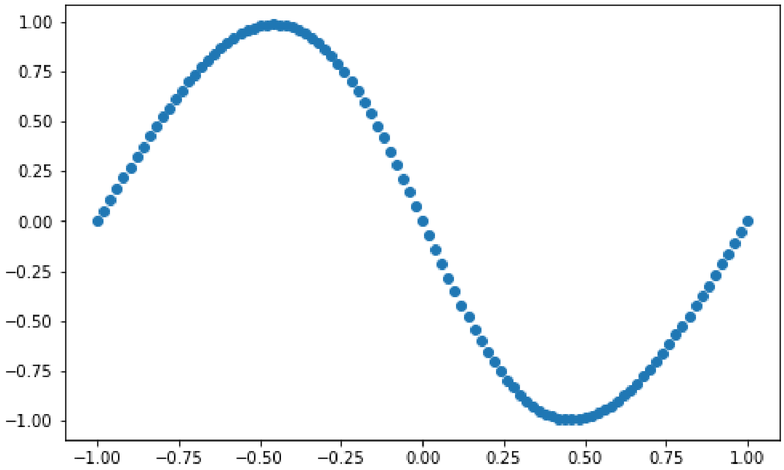
Код для нейронной сети писался с помощью открытой программной библиотеки tensorflow (версия 1) от компании Google.
Работа с объектами, в частности матричными, осуществлялась с помощью библиотеки numpy. В библиотеке numpy реализованы поддержка вычислительных алгоритмов для работы с многомерными массивави, что значительно экономит время и ресурсы при тренировке нейронных сетей. Также в numpy существует поддержка математических функций для работы с массивами.
Для визуализации результатов исследования использовалась библиотека matplotlib, предназначенная для построения 2D и 3D графиков. Средой разработки был выбран Jupyter Notebook - инструмент, где можно создовать отчеты, сочетающие в себе одновременно код, комментарии, тексты и графики.

0.6 Детали реализации

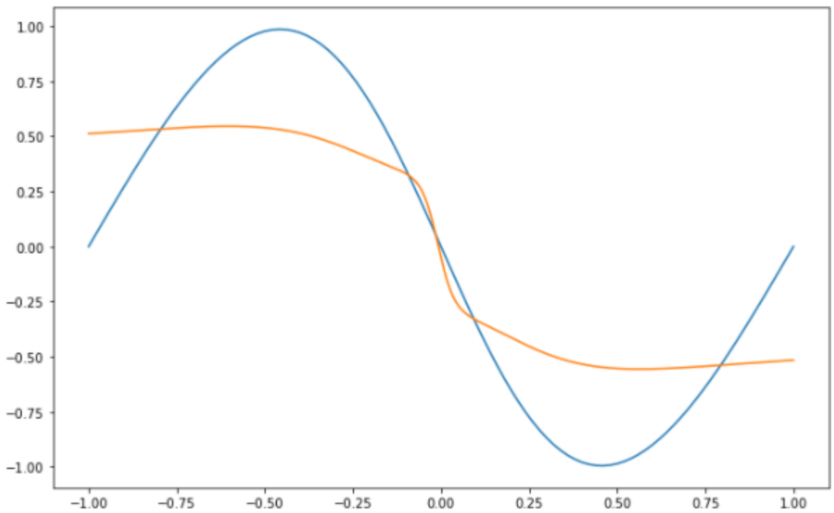
Нейронная сеть является полносвязной, с количеством слоев по умолчанию равным шести. Общая конструкция следующая - [2,10,10,10,10,10,1]. Здесь два - это количество входных нейронов, соответствующих вектору x(вектор состоит из точек из области определения x, то есть из интервала [-1,1]) и вектору t (состоит из точек области определения t, то есть из интервала [0,10]). Далее идут внутренние слои, каждый из которыз по 10 нейронов. Выходной нейрон соответствет вектору $u(t, x, \theta)$, являющимся значениями искомого решения. В качестве функции активации использовалась тангенсальная функция. Начальная инициализация весов реализована с помощью алгоритма xavier initialization. Предсказание финкции вида: $u_t + uu_x - \frac{0.01}{\pi} uu_{xx}$ выполнено при помощи функции tensorflow.gradient библиотеки tensorflow. В качестве метода оптимизации нейронной сети выбран алгоритм Adam.

0.7 Результаты

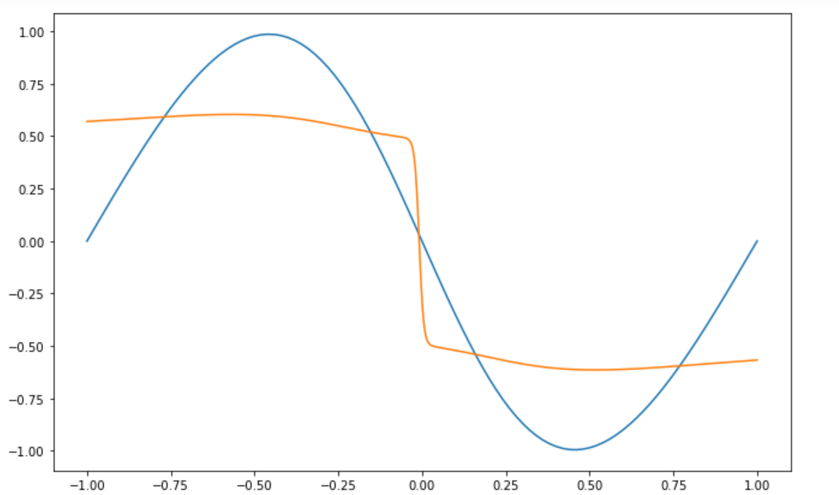
Численное решение уравнения на сетке:



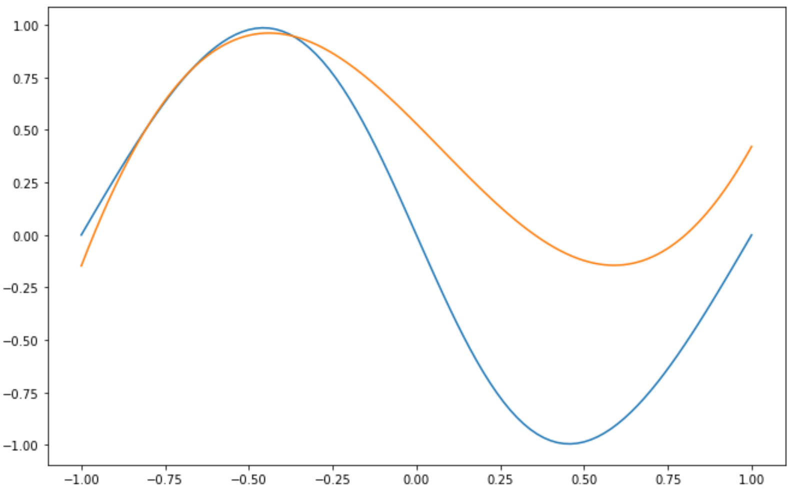
Результаты на искусственно сгенерированных данных. Оранжевый цвет - аппроксимация с помощью нейронной сети.
Количество итераций = 5000:



Результаты на искусственно сгенерированных данных. Оранжевый цвет - аппроксимация с помощью нейронной сети.
Количество итераций = 20000:



Результаты полученные на данных, взятых из сети и предназначавшихся для решения уравнения Навье-Стокса.
Количество итераций $= 5000$



0.8 Ссылки

1. "A deep learning algorithm for solving partial differentialequations"by Justin Sirignano
2. Tensorflow documintation: www.tensorflow.org
3. "A Discussion on Solving Partial DifferentialEquations using Neural Networks"by Tim Dockhorn
4. Burgers' equation - wikipedia.org/wiki/Burgers

5. "Neural Networks for Solving PDEs"by Anastasia Borovykh

6. "Machine Learning for Partial Differential Equations"by Michael Brenner

7. "Burgers solution": https://www.hindawi.com/journals/mpe/2015/