Внедрение оптимизационных алгоритмов в интерфейс фреймворка Streamlit[1]

Егоров П.А., СПбГУ, Санкт-Петербург st068715@student.spbu.ru

Аннотация

В ходе работы были реализованы на языке Python и интегрированы в приложение machine-learning-ui оптимизационные алгоритмы AdaMod, MADGRAD, RAdam, Apollo, AdaHessian, LARS, LAMB. В качестве апробации алгоритмов была решена задача расчета поуровневых коэффициентов скорости колебательных энергообменов с применением нейросетевого подхода.

Введение

Streamlit — это открытый Python фреймворк, который позволяет создавать интерактивные веб-приложения для анализа данных и машинного обучения. Фреймворк имеет понятный интуитивный интерфейс и совместим со множеством библиотек, такими как Pandas, Matplotlib, TensorFlow и другими.

Необходимой частью машинного обучения являются оптимизационные алгоритмы, поскольку они позволяет находить оптимальные параметры моделей, минимизировать функцию потерь и повышать точность предсказаний. В нейронных сетях, как правило, используют алгоритмы, основанные на вычислении градиента или гессиана оптимизируемой функции.

На языке Python были реализованы и интегрированы в приложение machine-learning-ui, разработанное на кафедре гидроаэромеханики в рамках исследовательского проекта «Машинное обучение в задачах неравновесной аэромеханики», следующие оптимизаторы: AdaMod [1], MADGRAD [2], RAdam [3], Apollo [4], AdaHessian [5], LARS [6], LAMB [7]. Основным инструментом для создания алгоритмов является библиотека TensorFlow. Для внедрения оптимизаторов в приложение были реализованы виджеты с учетом логики Streamlit, поскольку приложение создано с использованием данного фреймворка.

Детали реализации

В качестве основного инструмента для реализации оптимизационных алгоритмов была использована библиотека TensorFlow. Таким образом, все оптимизаторы должны наследоваться от базового класса

[1]M1_2021 - 3: Машинное обучение в задачах неравновесной аэромеханики: 2023 г. этап 3

`tensorflow.keras.optimizers.Optimizer`. При создании алгоритмов необходимо переопределить несколько методов базового класса. В конструкторе `__init__()` определяются гиперпараметры модели, такие как скорость обучения, коэффициенты затухания и т.д. Инициализация переменных оптимизатора, например, импульса или экспоненциального скользящего среднего градиентов осуществляется в методе `build()`, принимающем на вход список переменных модели `var_list`. Основная логика алгоритма реализуется в методе `update_step()`, который в качестве аргументов принимает переменную и градиент. Результат работы данного метода обновляет веса нейронной сети. Для сохранения конфигурации оптимизатора и последующего запуска реализуется метод `get_config()`, возвращающий словарь со значениями гиперпараметров оптимизационного алгоритма.

В оптимизаторе AdaHessian, помимо переменных и градиентов, которые автоматически вычисляются в TensorFlow, необходимо хранить информацию об аппроксимации матрицы Гессе, которая рассчитываются при дифференцировании градиентов функции потерь. Поэтому при реализации данного алгоритма были переопределены и другие методы класса `Optimizer`, включая такие, как `compute_gradients()`, `apply_gradients()`, `minimize()` и т.д.

Основное приложение написано с использованием фреймворка Streamlit. В связи с этим для интеграции оптимизационных алгоритмов были реализованы виджеты с учетом логики Streamlit. Классы виджетов содержат в себе описания, диапазоны значений и строковые форматы гиперпараметров оптимизаторов.

Диаграммы реализованных оптимизаторов выглядят следующим образом:

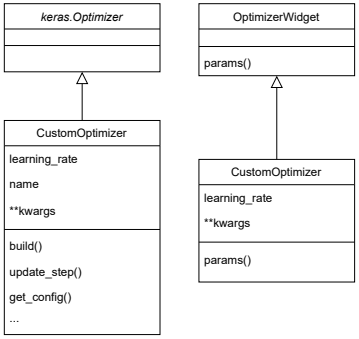

Рис. 1: Диаграммы классов оптимизаторов и виджетов

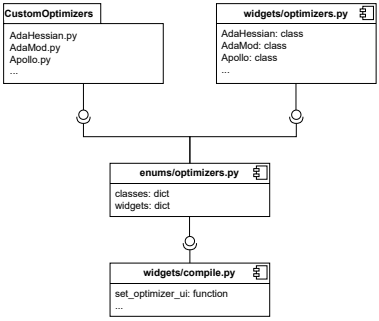

Рис. 2: Диаграмма компонентов

Тестирование алгоритмов

Условия эксперимента

Для тестирования работы оптимизационных алгоритмов с помощью библиотек NumPy и Pandas был сгенерирован набор данных, равномерно распределенных на отрезке [0, 1), размерности 10000×50 с добавлением шума, имеющего стандартное нормальное распределение. В качестве тестовой модели была использована нейронная сеть с тремя входными слоями, каждый из которых соединен с отдельным полносвязным слоем. Далее расположен слой конкатенации, связывающий три предыдущих слоя, а за ним еще один полносвязный слой. Он, в свою очередь, разделяется еще на два полносвязных слоя, которые являются выходными слоями модели.

Помимо обучения с использованием реализованных оптимизационных методов, в ходе эксперимента модель была обучена с оптимизатором Adam, доступным в библиотеке TensorFlow. В таблице 1 представлены гиперпараметры алгоритмов, использованные в процессе эксперимента.

Оптимизатор	Гиперпараметры
AdaMod	lr = 0,001, β_1 = 0,9, β_2 = 0,999, β_3 = 0,995
RAdam	lr = 0,001, β_1 = 0,9, β_2 = 0,999
MADGRAD	lr = 0,01, `momentum` = 0,9
Apollo	lr = 0,01, β = 0,9, `weight_decay` = 0,001
AdaHessian	lr = 0,01, β_1 = 0,9, β_2 = 0,999, `weight_decay` = 0,001
LARS	lr = 0,01, β = 0,9
LAMB	lr = 0,001, β_1 = 0,9, β_2 = 0,999
Adam	lr = 0,001, β_1 = 0,9, β_2 = 0,999

Таблица 1: Оптимизаторы и их гиперпараметры

Метрики

В качестве функции потерь использовалась среднеквадратичная ошибка (Mean Squared Error, MSE). Она имеет свойство штрафовать большие ошибки сильнее, нежели другие метрики за счет того, что результат ошибки возводится в квадрат.

Также для оценки качества обученных моделей была использована метрика — логарифм гиперболического косинуса ошибки предсказания. Данная

метрика менее чувствительна к выбросам, в отличие от среднеквадратичной и средней абсолютной ошибок, а исходные данные были зашумлены.

Результаты обучения

В таблице 2 представлены значения времени обучения модели с различными оптимизаторами.

Алгоритм	Время обучения, с	Число эпох
AdaMod	$15,8 \pm 1,5$	22
MADGRAD	$19,6 \pm 1,3$	24
RAdam	$17,9 \pm 0,4$	23
Apollo	$12,6 \pm 1,4$	17
AdaHessian	$24,1 \pm 0,6$	30
LARS	$21,5 \pm 0,4$	30
LAMB	$15,8 \pm 1,5$	21
Adam	$16,36 \pm 0,8$	21

Таблица 2: Время обучения алгоритмов

Быстрее всех обучилась модель с оптимизатором Apollo — приблизительно 12,5 секунд. Примерно одинаковое время обучения показали алгоритмы первого порядка AdaMod, LAMB и Adam — около 16 секунд. RAdam превзошел по скорости MADGRAD, который оказался одним из самых долгообучаемых среди алгоритмов первого порядка, за исключением оптимизатора LARS со средней скоростью обучения 21,5 секунд. Медленнее всех обучилась модель с оптимизатором AdaHessian — примерно 24 секунды.

Расчет поуровневых коэффициентов скорости колебательных энергообменов

В качестве апробации была решена задача расчета коэффициентов скорости колебательных энергообменов. Для решения данной задачи были предоставлены наборы данных VT-обменов между молекулой O_2 и частицами партнерами по столкновению (O_2, O, Ar), рассчитанные с помощью модели нагруженного гармонического осциллятора с учетом свободных вращений. Размерности наборов данных составили 525 элементов для всех пар частиц. Элементы наборов данных состоят из номеров уровней перехода, тем-

пературы и вычисленных коэффициентов. В работе [8] регрессионной моделью аппроксимировали коэффициенты отдельно для каждого уровня только по значениям температуры, но в этом случае размерность набора данных составляет всего 14 элементов. Данный подход нецелесообразно использовать при обучении нейронной сети. Поэтому нейросетевые модели были обучены по всему набору данных. Предсказание коэффициентов осуществляется по двум переменным — уровню перехода и температуре.

Для обучения была использована пятислойная модель нейронной сети. В первом и последнем полносвязных слоях использовалась тождественная функция активации для обработки выходного значения. В центральном полносвязном слое была применена сигмоидная функция активации, поскольку все данные больше нуля, а также для добавления нелинейности в модель. Для увеличения производительности и борьбы с переобучением между полносвязными слоями были добавлены слои пакетной нормализации (Batch Normalization Layer). В качестве функции потерь была выбрана среднеквадратичная ошибка.

Поскольку значения коэффициентов малы, для улучшения качества предсказаний они были предварительно нормализованы — умножены на 10×10^{18} и преобразованы до диапазона [0, 1] с помощью MinMaxScaler:

$$\hat{x_i} = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

где x_i - значение i-го элемента признака, x_{min} и x_{max} - минимальное и максимальное значения признака.

В таблице 3 представлены значения ошибок и времени обучения для всех наборов данных после масштабирования.

Алгоритм	O_2-O		O_2-Ar		O_2-O_2	
	MSE	Время, с	MSE	Время, с	MSE	Время, с
AdaMod	0,004	6,23	0,117	4,31	0,015	4,44
MADGRAD	0,223	4,76	0,256	4,68	0,168	4,57
RAdam	0,072	6,65	0,069	5,04	0,026	4,94
Apollo	0,261	2,48	0,300	2,85	0,174	2,46
AdaHessian	0,129	5,92	0,087	5,86	0,114	5,85
LARS	0,007	4,43	0,025	4,38	0,029	4,43
LAMB	0,027	4,47	0,063	4,78	0,036	4,67
Adam	0,100	4,23	0,338	3,11	0,352	3,42

Таблица 3: Результаты обучения на преобразованных данных

На рисунке 3 представлены предсказания коэффициентов моделью с оптимизатором AdaMod для задачи O_2-O_2 взаимодействий при одноквантовом переходе с 32 на 31 уровень.

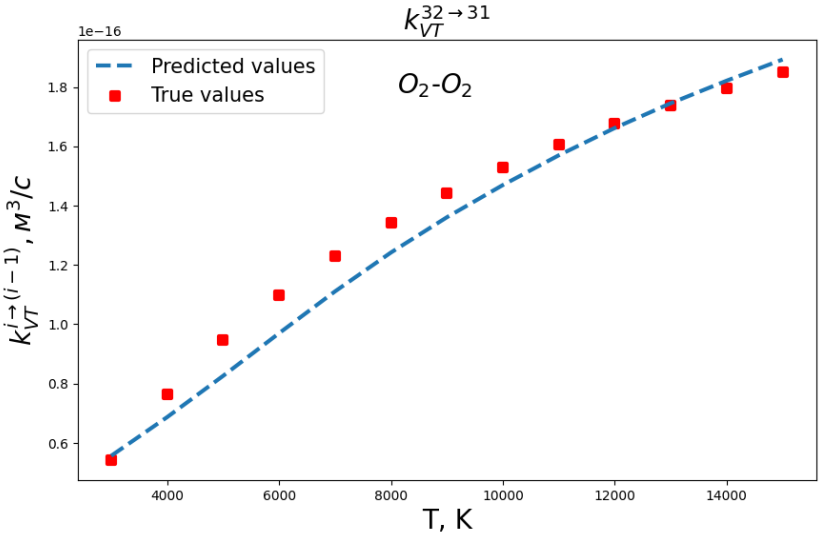

Рис. 3: $k_{VT}^{32\to31}$ O_2-O_2 взаимодействий

Заключение

В ходе работы на языке Python с использованием библиотеки TensorFlow были реализованы и интегрированы в приложение machine-learning-ui оптимизационные алгоритмы: AdaMod, RAdam, MADGRAD, Apollo, AdaHessian, LARS, LAMB. Была проведена апробация алгоритмов. Модель нейросети со всеми оптимизаторами сошлась в течение 30 эпох.

Также была решена задача расчета поуровневых коэффициентов скорости колебательных энергообменов для пар частиц (O_2-O, O_2-Ar, O_2-O_2) с применением нейросетевого подхода. Была подобрана подходящая топология модели для текущей задачи. Обучение модели с различными оптимизационными алгоритмами заняло менее 7 секунд для каждого набора данных с учетом масштабирования. Значение ошибки при валидации реализованных оптимизаторов составило от 0,3 до 0,004, а средняя ошибка после обратного масштабирования оказалась равной — $1,19 \times 10^{-17}$. Предсказанные значения оказались близки к истинным.

Список литературы

[1] Jianbang Ding, Xuancheng Ren, Ruixuan Luo, Xu Sun. An Adaptive and Momental Bound Method for Stochastic Learning. // arXiv:1910.12249, 2019

[2] Aaron Defazio, Samy Jelassi. Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization. // arXiv:2101.11075, 2021

[3] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. // arXiv:1908.03265, 2021

[4] Xuezhe Ma. Apollo: An Adaptive Parameter-wise Diagonal Quasi-Newton Method for Nonconvex Stochastic Optimization. // arXiv:2009.13586, 2021

[5] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, Michael W. Mahoney. ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning. // arXiv:2006.00719, 2021

[6] Yang You, Igor Gitman, Boris Ginsburg. Large Batch Training of Convolutional Networks. // arXiv:1708.03888, 2017

[7] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. // arXiv:1904.00962, 2020

[8] Исаков Андрей Алексеевич, Гориховский Вячеслав Игоревич, Мельник Максим Юрьевич. Модели регрессии для расчёта поуровневых коэффициентов скорости колебательных энергообменов. // ВЕСТНИК САНКТ-ПЕТЕРБУРГСКОГО УНИВЕРСИТЕТА. МАТЕМАТИКА. МЕХАНИКА. АСТРОНОМИЯ, 2024