Санкт-Петербургский государственный Университет
Математико-Механический факультет
Кафедра системного программирования

Отчет по научно-исследовательской работе
Тема: “Оптимизационные алгоритмы для нейросетевых гидроаэромеханических расчётов”

Выполнил:
студент группы 22.М04
Егоров П. А.
Научный руководитель:
к.ф.-м.н. Гориховский В. И.

Санкт-Петербург
2024

1. Введение

Машинное обучение и нейронные сети стали неотъемлемой частью современной науки. Они обеспечивают возможность не только анализировать огромные объемы данных, но и выявлять скрытые закономерности и шаблоны в этих данных. Это важно, поскольку данные, часто слишком сложные для человека, могут быть эффективно обработаны и проанализированы компьютерами с использованием методов машинного обучения.

Применение методов машинного обучения и нейронных сетей позволяет решать разнообразные сложные задачи, включая задачи неравновесной аэромеханики и газодинамики. В контексте этих дисциплин, машинное обучение позволяет автоматизировать процессы анализа данных, прогнозирование результатов и создание высокоэффективных моделей, что в свою очередь улучшает понимание процессов и явлений, которые ранее могли быть сложными для восприятия или анализа.

Например, в задачах неравновесной аэромеханики, машинное обучение и нейронные сети могут использоваться для моделирования и анализа потоков воздуха вокруг объектов, таких как самолеты или автомобили. Это позволяет оптимизировать форму и конструкцию объектов для улучшения их аэродинамических характеристик и повышения эффективности.

В задачах газодинамики, машинное обучение и нейронные сети могут помочь в анализе и прогнозировании поведения газовых сред, таких как атмосфера или газовые потоки в трубопроводах, что может быть полезно, например, при прогнозировании погоды.

Важным аспектом машинного обучения и нейронных сетей является применение оптимизационных алгоритмов. Это необходимо при решении различных типов задач с использованием нейронных сетей, поскольку оптимизаторы позволяют настраивать параметры модели таким образом, чтобы минимизировать функцию потерь. Это позволяет улучшить производительность моделей и повысить точность их прогнозирования.

Существуют различные подходы для решения задач оптимизации. В нейронных сетях, как правило, используют алгоритмы, основанные на вычислении градиента или гессиана оптимизируемой функции. К алгоритмам первого порядка, которые используют градиент для вычисления локального минимума, можно отнести стохастический градиентный спуск[1], Adam[2] и т.д. Методы второго порядка для оптимизации используют матрицу Гессе, также к ним относятся квазиньютоновские алгоритмы, основанные на приближённых выражениях для матрицы Гессе, например, BFGS[3].

Была поставлена задача добавления различных оптимизационных алгоритмов в приложение для решения задач гидроаэромеханики в рамках исследовательского проекта, финансируемого Санкт-Петербургским государственным университетом.

2. Постановка задачи

Приложение написано на языке Python с использованием фреймворка Streamlit. В качестве инструмента для создания и обучения нейросетевых моделей используется библиотека TensorFlow. Таким образом, реализованные оптимизаторы должны быть подклассами базового класса Optimizer библиотеки TensorFlow. Их интеграция в приложение должна учитывать специфику фреймворка Streamlit.

Предполагается использовать приложение для задач предсказания - классификации и регрессии. Однако, если в будущем возникнет необходимость решать другие типы задач, например распознавания, то также потребуется использование оптимизационных алгоритмов для эффективной настройки параметров модели и достижения оптимальных результатов.

Целью данной работы является применение различных оптимизационных схем: первого и второго порядков, а также быстро-обучаемых оптимизаторов для нахождения оптимальных вычислительных конфигураций с заданным уровнем точности проводимых расчетов.

3. Тестовая модель и тестовый датасет

В качестве тестовой модели для оценки работы оптимизаторов была использована трехслойная модель нейросети с тремя входными и двумя выходными слоями (Рис. 1). Входной слой является вспомогательным и нужен для того, чтобы устанавливать размер входного тензора. Выходной слой представляет собой последний слой, который выдает результат работы нейронной сети. В задачах классификации выходной слой может содержать вероятности принадлежности объекта к каждому из классов, а в задачах регрессии - числовое значение, которое предсказывает модель.

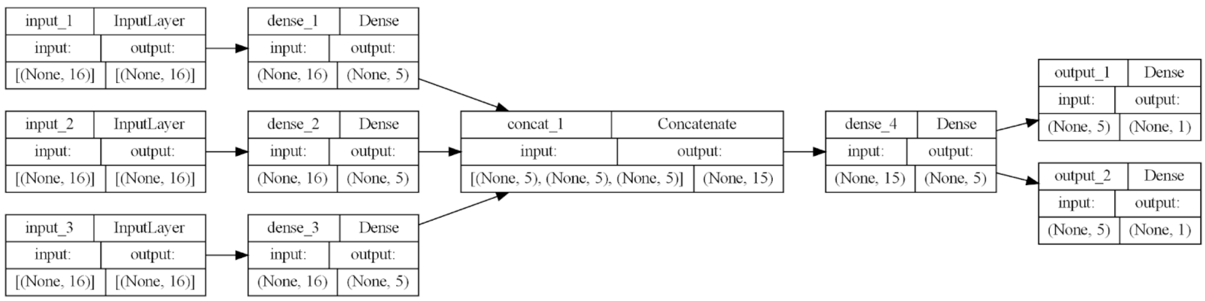

Рис. 1 Модель тестовой нейронной сети

Тестовый датасет состоял из пятидесяти признаков - по шестнадцать на каждый входной слой и по одному на каждый выходной.

Во время всех экспериментов в качестве функций потерь использовалась средняя абсолютная ошибка (MAE)[4], а для метрик модели использовалась среднеквадратичная ошибка (MSE)[5]. Выборка делилась на тестовую и обучающую в соотношением один к четырем. Обучение проводилось на тридцати эпохах.

4. Реализованные оптимизационные алгоритмы

4.1 An Adaptive and Momental Bound Method

Идея оптимизационного алгоритма AdaMod[6] состоит в том, чтобы ограничить скорость обучения адаптивными и моментальными верхними границами. Границы динамической скорости обучения основаны на экспоненциальных скользящих средних самих скоростей адаптивного обучения, которые сглаживают неожиданно высокие скорости обучения и стабилизируют обучение нейронных сетей. Это позволяет поддерживать баланс между скоростью сходимости и возможностью преодолеть локальные минимумы. Алгоритм AdaMod приведен на Рис. 2.

Algorithm 2 AdaMod

Input: initial parameter θ_0, step sizes $\{\alpha_t\}_{t=1}^{T}$, moment decay $\{\beta_1, \beta_2, \beta_3\}$, regularization constant ϵ, stochastic objective function $f(\theta_0)$

Initialize $m_0 = 0$, $v_0 = 0$, $s_0 = 0$
for $t = 1$ **to** T **do**
  $g_t = \nabla f_t(\theta_{t-1})$
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t = m_t / (1 - \beta_1^t)$
  $\hat{v}_t = v_t / (1 - \beta_2^t)$
  $\eta_t = \alpha_t / (\sqrt{\hat{v}_t} + \epsilon)$
  $s_t = \beta_3 s_{t-1} + (1 - \beta_3) \eta_t$
  $\hat{\eta}_t = \min(\eta_t, s_t)$
  $\theta_t = \theta_{t-1} - \hat{\eta}_t \hat{m}_t$
end for

Рис. 2 Алгоритм AdaMod

На Рис. 3 представлены результаты обучения модели с оптимизатором AdaMod со следующими параметрами: скорость обучения - 0.001, коэффициенты, используемые для вычисления средних значений градиента и его квадрата - 0.9 и 0.999, коэффициент сглаживания для адаптивных скоростей обучения - 0.995.

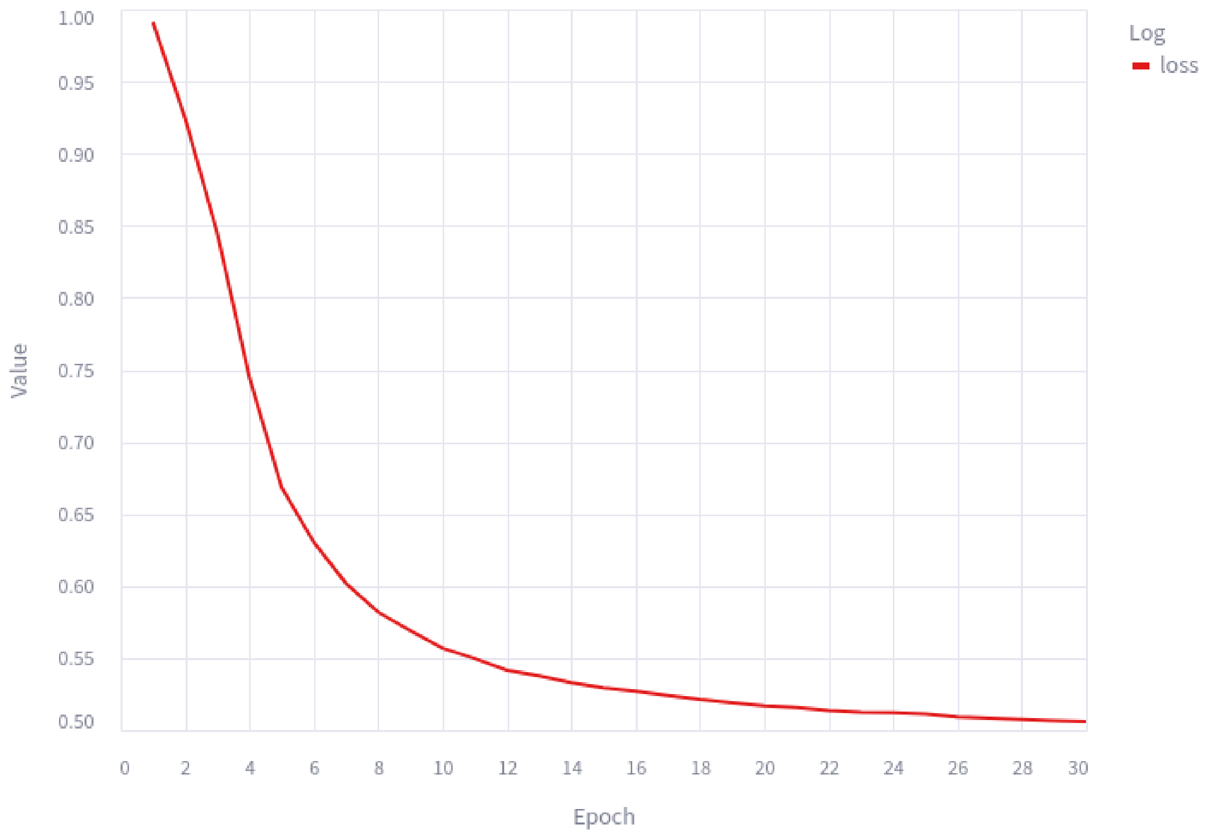

Рис. 3 Результат обучения модели с использованием алгоритма AdaMod

4.2 An Adaptive Parameter-wise Diagonal Quasi-Newton Method

Apollo - это квазиньютоновский метод невыпуклой стохастической оптимизации, который динамически учитывает кривизну функции потерь путем аппроксимации гессиана диагональной матрицей[7]. Обновление и хранение диагональной аппроксимации гессиана так же эффективно, как и адаптивные методы оптимизации первого порядка с линейной сложностью как по времени, так и по памяти. Чтобы справиться с невыпуклостью гессиан заменяется его выпрямленным абсолютным значением, которое гарантированно будет положительно определенным. Алгоритм Apollo приведен на Рис. 4.

Algorithm 2: APOLLO with weight decay (L_2/Decoupled)

Initial: $m_0, d_0, B_0 \leftarrow 0, 0, 0$ // Initialize m_0, d_0, B_0 to zero

while $t \in \{0, \dots, T\}$ **do**

  for $\theta \in \{\theta^1, \dots, \theta^L\}$ **do**

    $g_{t+1} \leftarrow \nabla f_t(\theta_t) + \gamma\theta_t$ // Calculate gradient at step t

    $m_{t+1} \leftarrow \frac{\beta(1-\beta^t)}{1-\beta^{t+1}} m_t + \frac{1-\beta}{1-\beta^{t+1}} g_{t+1}$ // Update bias-corrected moving

    $\alpha \leftarrow \frac{d_t^T(m_{t+1}-m_t) + d_t^T B_t d_t}{(\|d_t\|_4 + \epsilon)^4}$ // Calculate coefficient of B update

    $B_{t+1} \leftarrow B_t - \alpha \cdot \mathrm{Diag}(d_t^2)$ // Update diagonal Hessian

    $D_{t+1} \leftarrow \mathrm{rectify}(B_{t+1}, 0.01)$ // Handle nonconvexity

    $d_{t+1} \leftarrow D_{t+1}^{-1} m_{t+1} + \gamma\theta_t$ // Calculate update direction

    $\theta_{t+1} \leftarrow \theta_t - \eta_{t+1} d_{t+1}$ // Update parameters

  end

end

return θ_T

Рис. 4 Алгоритм Apollo

На Рис. 5 представлены результаты обучения модели с оптимизатором Apollo со следующими параметрами: скорость обучения - 0.01, коэффициент, используемый для вычисления средних значений градиента - 0.9, коэффициент затухания веса - 0.

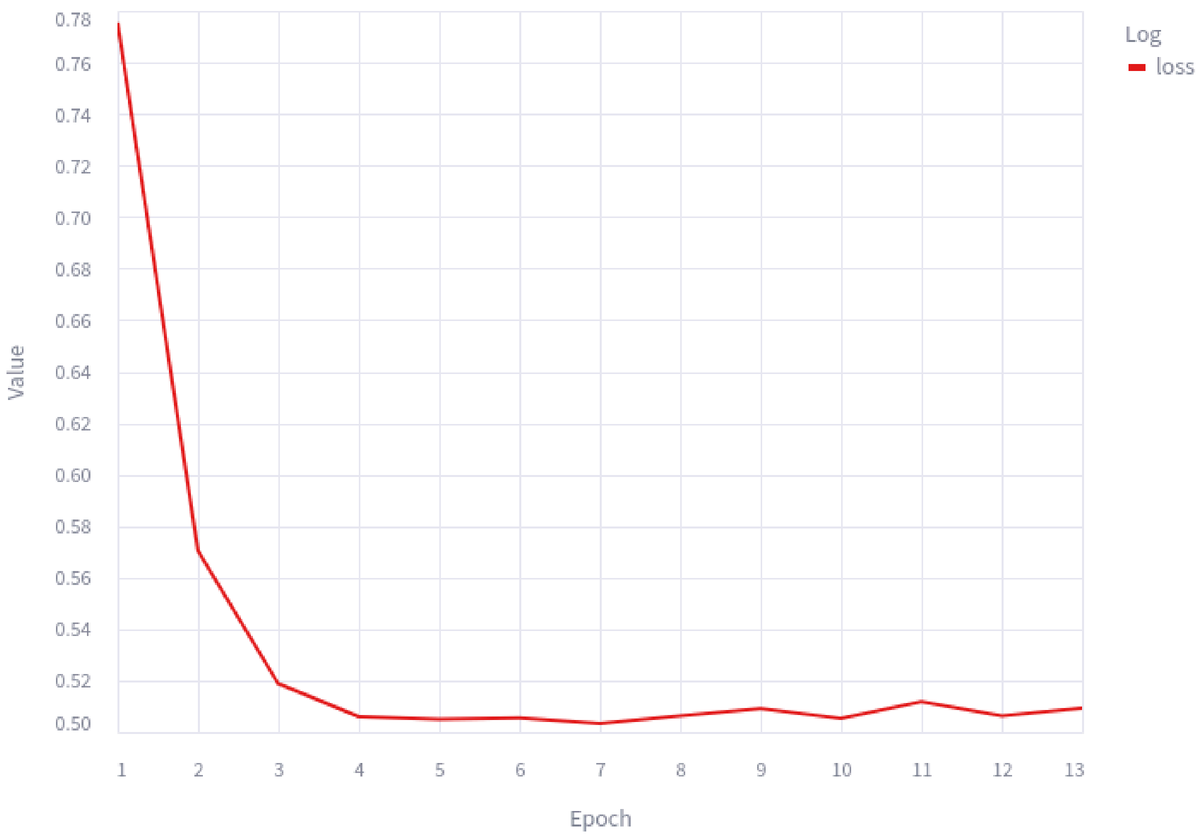

Рис. 5 Результат обучения модели с использованием алгоритма Apollo

4.3 Layer-wise Adaptive Rate Scaling

LARS - это алгоритм оптимизации, который автоматически адаптирует скорость обучения для каждого слоя модели[8]. Он основан на идее, что более глубокие слои модели могут иметь различные скорости обучения, поскольку они могут иметь различную чувствительность к изменению входных данных. Оптимизатор LARS может быть особенно полезен при обучении глубоких нейронных сетей, где каждый слой может иметь различную чувствительность к изменению входных данных. В качестве базового оптимизатора, так же как и в оригинальной статье, используется стохастический градиентный спуск (SGD). Алгоритм LARS приведен на Рис. 6.

Algorithm 1 SGD with LARS. Example with weight decay, momentum and polynomial LR decay.

Parameters: base LR γ_0, momentum m, weight decay β, LARS coefficient η, number of steps T
Init: $t = 0, v = 0$. Init weight w_0^l for each layer l
while $t < T$ for each layer l **do**
  $g_t^l \leftarrow \nabla L(w_t^l)$ (obtain a stochastic gradient for the current mini-batch)
  $\gamma_t \leftarrow \gamma_0 * \left(1 - \frac{t}{T}\right)^2$ (compute the global learning rate)
  $\lambda^l \leftarrow \frac{||w_t^l||}{||g_t^l|| + \beta ||w_t^l||}$ (compute the local LR λ^l)
  $v_{t+1}^l \leftarrow m v_t^l + \gamma_{t+1} * \lambda^l * (g_t^l + \beta w_t^l)$ (update the momentum)
  $w_{t+1}^l \leftarrow w_t^l - v_{t+1}^l$ (update the weights)
end while

Рис. 6 Алгоритм LARS

На Рис. 7 представлены результаты обучения модели с оптимизатором LARS со следующими параметрами: скорость обучения - 0.01, импульс - 0.9, коэффициент затухания веса - 0.01.

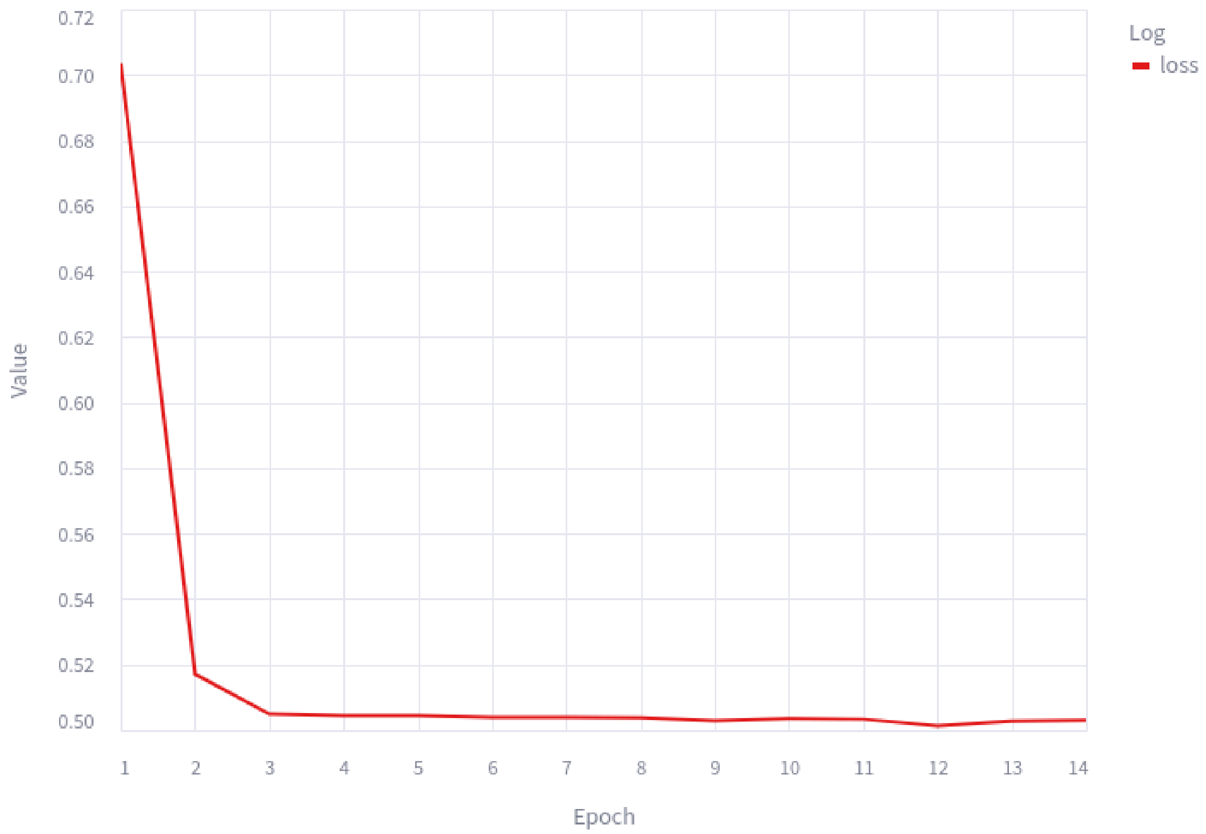

Рис. 7 Результат обучения модели с использованием алгоритма LARS

4.4 A Momentumized, Adaptive, Dual Averaged Gradient Method

MADGRAD[9] является оптимизационным алгоритмом на основе градиентного спуска первого порядка. Он показал неплохие результаты по сравнению с классическими SGD и Adam. Также MADGRAD менее чувствителен к подбору гиперпараметров. Алгоритм MADGRAD приведен на Рис. 8.

Algorithm 1 MADGRAD

Require: γ_k stepsize sequence, c_k momentum sequence, initial point x_0, epsilon ϵ

$s_0 : d = 0$, $\nu_0 : d = 0$

for $k = 0, \ldots, T$ **do**

  Sample ξ_k and set $g_k = \nabla f(x_k, \xi_k)$

  $\lambda_k = \gamma_k \sqrt{k+1}$

  $s_{k+1} = s_k + \lambda_k g_k$

  $\nu_{k+1} = \nu_k + \lambda_k (g_k \circ g_k)$

$$z_{k+1} = x_0 - \frac{1}{\sqrt[3]{\nu_{k+1}} + \epsilon} \circ s_{k+1}$$

  $x_{k+1} = (1 - c_{k+1}) x_k + c_{k+1} z_{k+1}.$

end for

return x_T

Рис. 8 Алгоритм MADGRAD

На Рис. 9 представлены результаты обучения модели с оптимизатором MADGRAD со следующими параметрами: скорость обучения - 0.01, импульс - 0, коэффициент затухания веса - 0.

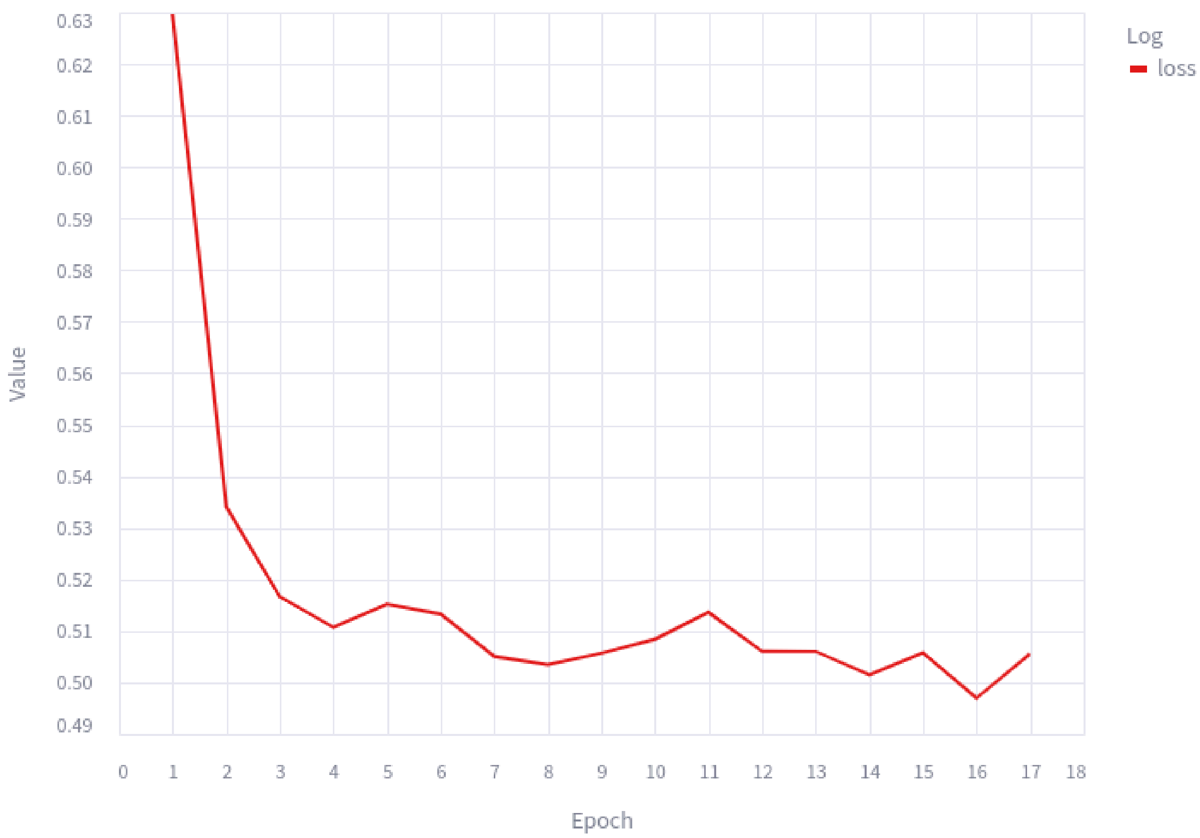

Рис. 9 Результат обучения модели с использованием алгоритма MADGRAD

4.5 An Adaptive Second Order Optimizer

AdaHessian - это оптимизационный алгоритм второго порядка в основе которого лежат следующие идеи[10]:

- диагональная аппроксимация гессиана с помощью метода Хатчинсона.
- среднеквадратическое экспоненциальное скользящее среднее для сглаживания изменений диагонали гессиана на разных итерациях.
- блочное диагональное усреднение для уменьшения дисперсии диагональных элементов Гессе.

На рисунке 10 представлен алогритм AdaHessian.

Algorithm 1: AdaHessian

Require: Initial Parameter: θ_0
Require: Learning rate: η
Require: Exponential decay rates: β_1, β_2
Require: Block size: b
Require: Hessian Power: k
Set: $m_0 = 0$, $v_0 = 0$
for $t = 1, 2, \ldots$ **do** // Training Iterations
  $\mathbf{g}_t \leftarrow$ current step gradient
  $\boldsymbol{D}_t \leftarrow$ current step estimated diagonal Hessian
  Compute $\boldsymbol{D}_t^{(s)}$ based on Eq. 10
  Update $\bar{\boldsymbol{D}}_t$ based on Eq. 11
  Update m_t, v_t based on Eq. 12
  $\theta_t = \theta_{t-1} - \eta m_t / v_t$

Рис. 10 Алгоритм AdaHessian

На Рис. 11 представлены результаты обучения модели с оптимизатором AdaHessian со следующими параметрами: скорость обучения - 0.15, коэффициенты, используемые для вычисления средних значений градиента и квадрата следа гессиан - 0.9 и 0.999.

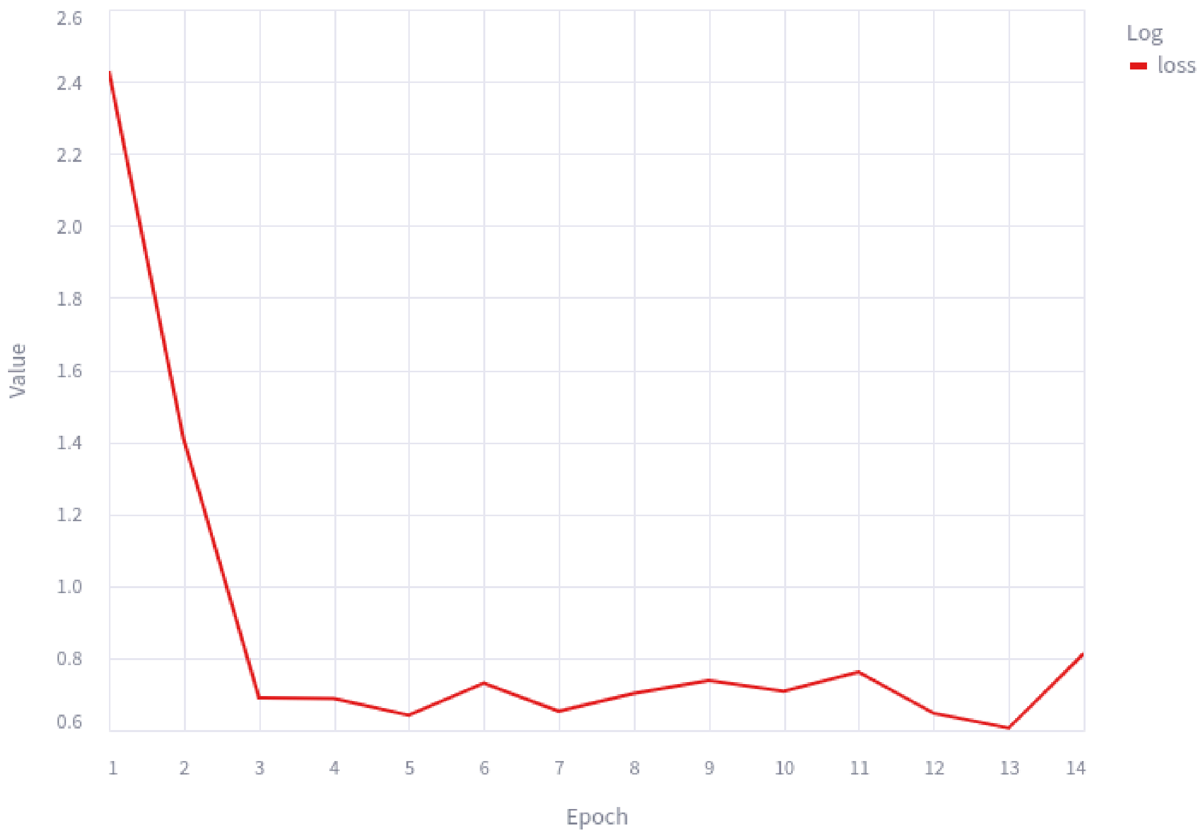

Рис. 11 Результат обучения модели с использованием алгоритма AdaHessian

5. Встроенные оптимизационные алгоритмы

Помимо реализованных вручную алгоритмов оптимизации, в приложении можно использовать алгоритмы из библиотеки Tensorflow. Рассмотрим следующие алгоритмы: стохастический градиентный спуск(SGD)[1], Root Mean Square Propagation(RMSProp)[11] и Adaptive Moment Estimation(Adam)[2].

Результаты обучения модели с использованием данных оптимизационных схем представлены на Рис.12, Рис. 13 и Рис. 14.

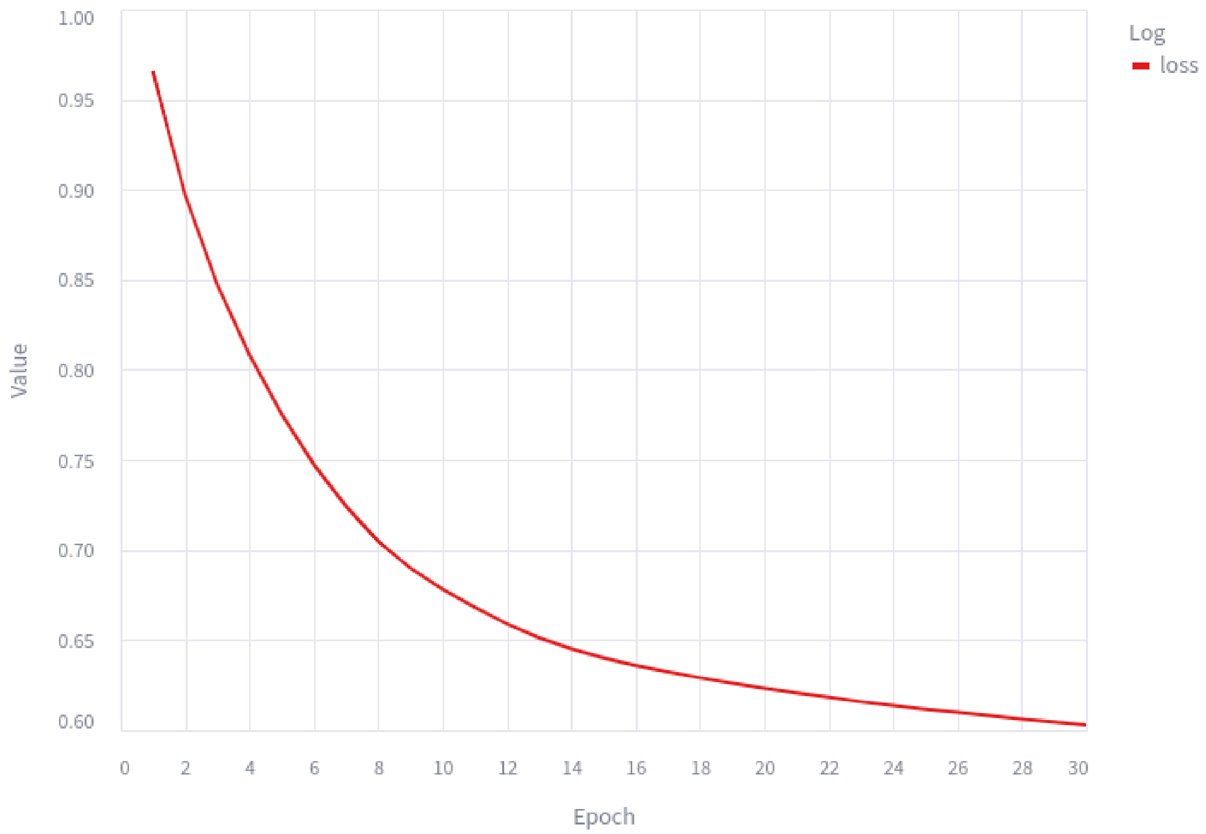

Рис. 12 Результат обучения модели с использованием алгоритма SGD

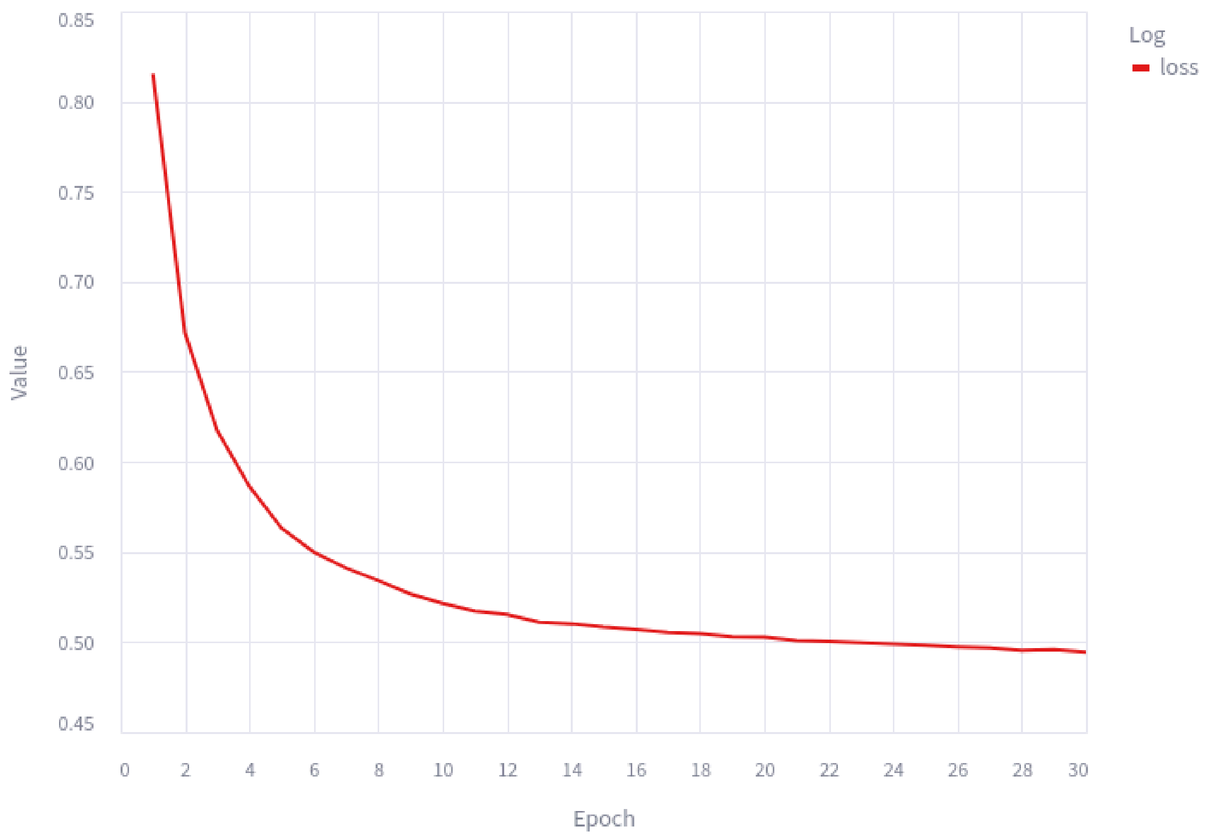

Рис. 13 Результат обучения модели с использованием алгоритма RMSProp

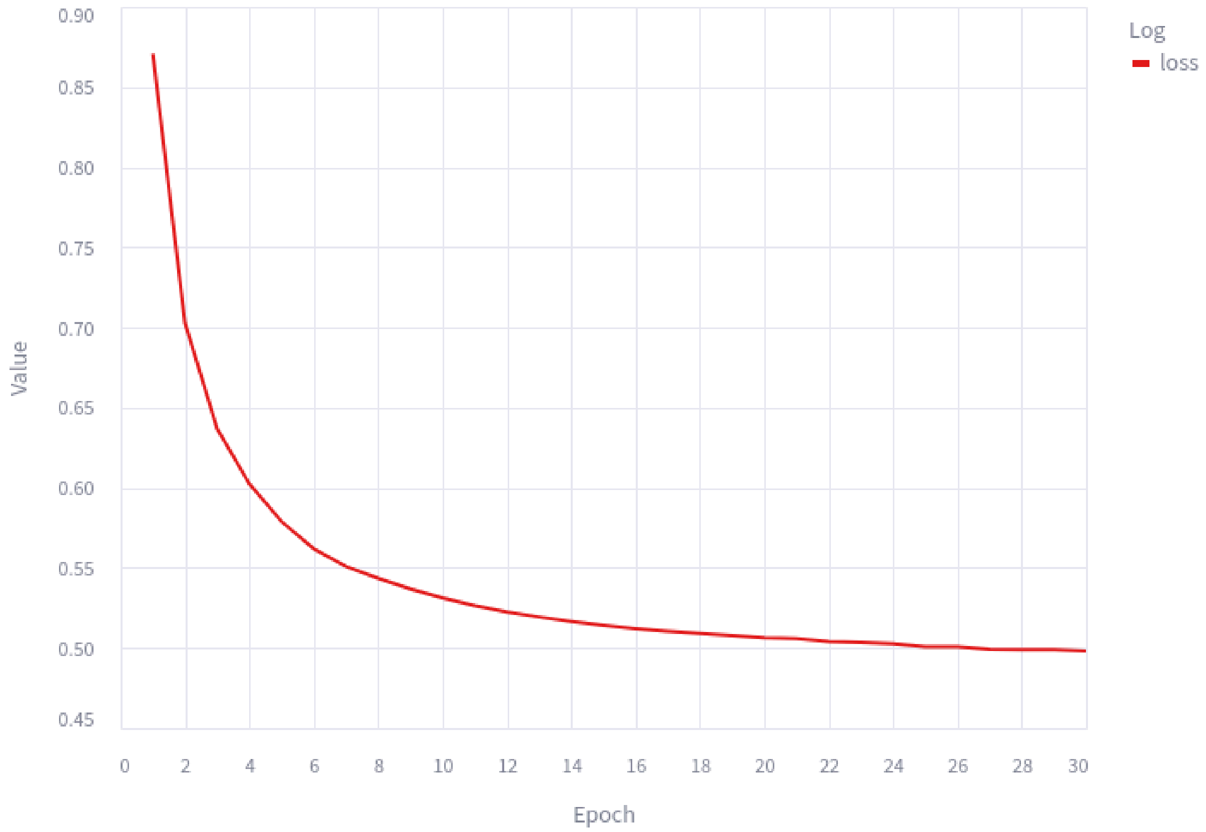

Рис. 14 Результат обучения модели с использованием алгоритма Adam

6. Заключение

В ходе учебной практики были реализованы различные оптимизационные алгоритмы: AdaMod, Apollo, LARS, MADGRAD, AdaHessian.

Все алгоритмы были интегрированы в приложение и протестированы.

7. Список литературы

1. Matt Taddy. Stochastic Gradient Descent // Business Data Science: Combining Machine Learning and Economics to Optimize, Automate, and Accelerate Business Decisions. — New York: McGraw-Hill, 2019. — ISBN 978-1-260-45277-8.

2. Diederik P. Kingma, Jimmy Ba (2014), Adam: A Method for Stochastic Optimization, arXiv:1412.6980

3. Fletcher, Roger (1987), Practical Methods of Optimization (2nd ed.), New York: John Wiley & Sons, ISBN 978-0-471-91547-8.

4.Willmott, Cort J.; Matsuura, Kenji (December 19, 2005). "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". Climate Research. 30: 79–82. doi:10.3354/cr030079

5. "Mean Squared Error (MSE)". www.probabilitycourse.com. Retrieved 2020-09-12.

6. Jianbang Ding, Xuancheng Ren, Ruixuan Luo, Xu Sun (2019), An Adaptive and Momental Bound Method for Stochastic Learning, arXiv:1910.12249v1.

7. Xuezhe Ma (2021), Apollo: An Adaptive Parameter-wise Diagonal Quasi-Newton Method for Nonconvex Stochastic Optimization, arXiv:2009.13586v6.

8. Yang You, Igor Gitman, Boris Ginsburg (2017), Large Batch Training of Convolutional Networks, arXiv:1708.03888v3.

9. Aaron Defazio, Samy Jelassi (2021), Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization, arXiv:2101.11075v3.

10. Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, Michael W. Mahoney (2021), ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning, arXiv:2006.00719.

10. Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, Jiawei Han (2021), On the Variance of the Adaptive Learning Rate and Beyond, arXiv:1908.03265v4.

11. Yann N. Dauphin, Harm de Vries, Yoshua Bengio (2015), Equilibrated adaptive learning rates for non-convex optimization, arXiv:1502.04390.