Санкт-Петербургский государственный университет

Оптимизационные алгоритмы для нейросетевых гидроаэромеханических расчётов

Егоров Павел Алексеевич

Научный руководитель:
к.ф.-м.н. Гориховский В. И.

Санкт-Петербург
2024

Понятие оптимизатора нейронной сети

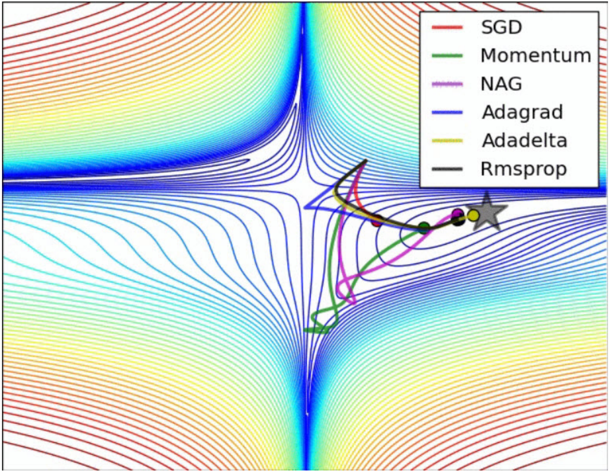

Рис. 1: Принцип работы оптимизационных алгоритмов

Цель работы

- Цель работы: применение различных оптимизационных схем: первого и второго порядков, а также быстрообучаемых оптимизаторов для нахождения оптимальных вычислительных конфигураций с заданным уровнем точности проводимых расчетов
- Используемые технологии: python 3.10, TensorFlow 2.12

Тестовая модель

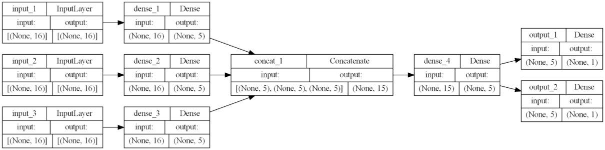

Рис. 2: Тестовая модель

- В качестве функций потерь моделей использовалась средняя абсолютная ошибка
- Выборка делилась на тестовую и обучающую в соотношением один к четырем
- Обучение проводилось на тридцати эпохах

Реализованные алгоритмы

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

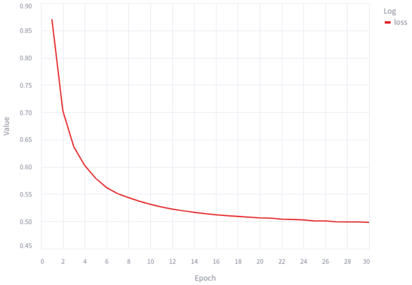

Рис. 3: Adamod

Реализованные алгоритмы

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 $k = 0, \dots, T$ **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

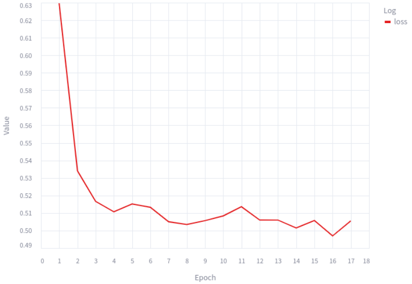

Рис. 4: MADGRAD

Реализованные алгоритмы

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

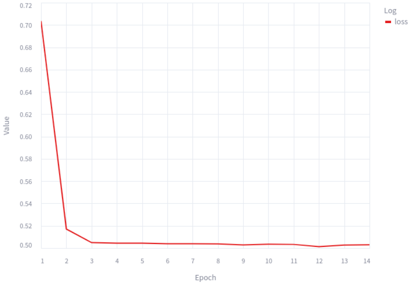

Рис. 5: LARS

Реализованные алгоритмы

Algorithm 1: APOLLO, our proposed algorithm for nonconvex stochastic optimization. All operations on vectors are element-wise. Good default settings are $\beta = 0.9$ and $\epsilon = 1e^{-4}$.

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**
    $g_{t+1} \leftarrow \nabla f_t(\theta_t)$ // 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 average
    $\alpha \leftarrow \frac{d_t^T(m_{t+1}-m_t)+d_t^T B_t d_t}{(\|d_t\|_4+\epsilon)^4}$ // 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
    $d_{t+1} \leftarrow D_{t+1}^{-1} m_{t+1}$ // Calculate update direction
    $\theta_{t+1} \leftarrow \theta_t - \eta_{t+1} d_{t+1}$ // Update parameters
  end
end
return θ_T

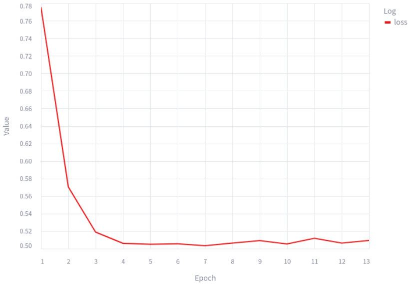

Рис. 6: Apollo

Реализованные алгоритмы

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
  $\theta_t = \theta_{t-1} - \eta m_t / v_t$

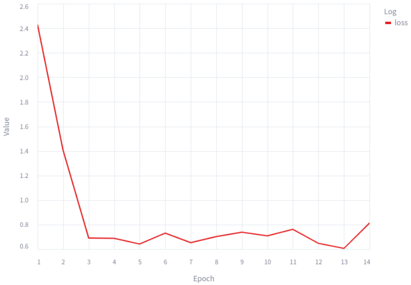

Рис. 7: AdaHessian

Встроенные алгоритмы

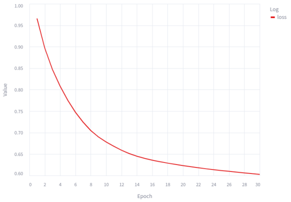

Рис. 8: SGD

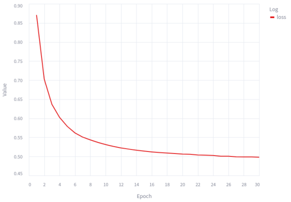

Рис. 9: Adam

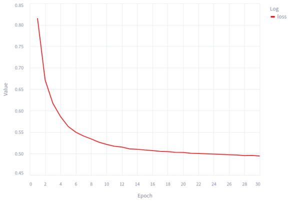

Рис. 10: RMSProp

Заключение

В ходе учебной практики были реализованы различные оптимизационные алгоритмы:

1. AdaMod
2. Apollo
3. LARS
4. MADGRAD
5. AdaHessian

Все алгоритмы были интегрированы в приложение и протестированы.