Методы матричной оптимизации на основе ортогонализации. Shampoo, Muon.

Даня Меркулов

ФКН ВШЭ

$$x_{k+1} = x_k - \alpha_k \cdot g_k$$

g_k – стох. градиент

Адаптивные стохастические методы

$$\min_{x \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

ADAM

Почему нужна адаптивность

- Единый скаляр α — плохой выбор, когда **разные координаты** имеют существенно разные масштабы:

Почему нужна адаптивность

- Единый скаляр α — плохой выбор, когда **разные координаты** имеют существенно разные масштабы:
  - редкие признаки в логистической регрессии и NLP-моделях;

Почему нужна адаптивность

- Единый скаляр α — плохой выбор, когда **разные координаты** имеют существенно разные масштабы:
  - редкие признаки в логистической регрессии и NLP-моделях;
  - в embedding-слое строка редкого слова получает ненулевой градиент только когда это слово попало в батч — в отличие от плотных слоёв, где все параметры обновляются на каждом шаге.

Почему нужна адаптивность

- Единый скаляр α — плохой выбор, когда **разные координаты** имеют существенно разные масштабы:
  - редкие признаки в логистической регрессии и NLP-моделях;
  - в embedding-слое строка редкого слова получает ненулевой градиент только когда это слово попало в батч — в отличие от плотных слоёв, где все параметры обновляются на каждом шаге.
- Идея адаптивных методов — **подобрать свой эффективный шаг каждой координате**, опираясь на накопленную историю градиентов.

Почему нужна адаптивность

- Единый скаляр α — плохой выбор, когда **разные координаты** имеют существенно разные масштабы:
  - редкие признаки в логистической регрессии и NLP-моделях;
  - в embedding-слое строка редкого слова получает ненулевой градиент только когда это слово попало в батч — в отличие от плотных слоёв, где все параметры обновляются на каждом шаге.
- Идея адаптивных методов — **подобрать свой эффективный шаг каждой координате**, опираясь на накопленную историю градиентов.
- Содержательно это можно трактовать как грубое приближение диагонали гессиана (или информационной матрицы Фишера).

Adagrad [1]

Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Обновление для каждой координаты $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{(k-1)} + \left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- Не требует тонкой настройки α — эффективный шаг убывает автоматически по каждой координате.

Adagrad [1]

Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Обновление для каждой координаты $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{(k-1)} + \left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- Не требует тонкой настройки α — эффективный шаг убывает автоматически по каждой координате.
- Для редких, но информативных признаков шаг убывает медленно — алгоритм даёт им «работать» дольше.

Adagrad [1]

Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Обновление для каждой координаты $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{(k-1)} + \left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- Не требует тонкой настройки α — эффективный шаг убывает автоматически по каждой координате.
- Для редких, но информативных признаков шаг убывает медленно — алгоритм даёт им «работать» дольше.
- Заметно улучшает SGD в разреженных задачах (NLP, рекомендательные системы).

Adagrad [1]

Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Обновление для каждой координаты $j = 1, \dots, p$:

$$v_j^{(k)} = v_j^{(k-1)} + \left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- Не требует тонкой настройки α — эффективный шаг убывает автоматически по каждой координате.
- Для редких, но информативных признаков шаг убывает медленно — алгоритм даёт им «работать» дольше.
- Заметно улучшает SGD в разреженных задачах (NLP, рекомендательные системы).
- Главная слабость — **монотонное** накопление в знаменателе: эффективный шаг $\to 0$ слишком рано, и обучение «затухает».

[1] Duchi, Hazan, Singer (2010). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.

RMSProp [2]

Решает проблему монотонно убывающей скорости обучения. Использует экспоненциально затухающее среднее квадратов градиентов:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)\left(g_j^{(k)}\right)^2$$

EMA

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- В знаменателе — скользящее среднее квадратов градиентов (RMS): «забывает» старые градиенты, в отличие от монотонного накопления в AdaGrad.

RMSProp [2]

Решает проблему монотонно убывающей скорости обучения. Использует экспоненциально затухающее среднее квадратов градиентов:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)\left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- В знаменателе — скользящее среднее квадратов градиентов (RMS): «забывает» старые градиенты, в отличие от монотонного накопления в AdaGrad.
- Эффективный шаг может как убывать, так и возрастать — подходит для нестационарных задач.

RMSProp [2]

Решает проблему монотонно убывающей скорости обучения. Использует экспоненциально затухающее среднее квадратов градиентов:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)\left(g_j^{(k)}\right)^2$$

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{g_j^{(k)}}{\sqrt{v_j^{(k)} + \varepsilon}}$$

Замечания:

- В знаменателе — скользящее среднее квадратов градиентов (RMS): «забывает» старые градиенты, в отличие от монотонного накопления в AdaGrad.
- Эффективный шаг может как убывать, так и возрастать — подходит для нестационарных задач.
- Долгое время был выбором по умолчанию при обучении рекуррентных сетей.

[2] Tieleman, Hinton (2012). Coursera Lecture 6.

Adadelta [3]

Развитие RMSProp: вообще **не требует** глобального шага α. Вместо него масштабирует шаг через накопленные обновления параметров:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)\left(g_j^{(k)}\right)^2$$

EMA

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \varepsilon}}{\sqrt{v_j^{(k)} + \varepsilon}}\, g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)},$$

$$\Delta x_j^{(k)} = \rho\, \Delta x_j^{(k-1)} + (1-\rho)\left(\tilde{g}_j^{(k)}\right)^2$$

EMA

Замечания:

- Числитель $\sqrt{\Delta x_j^{(k-1)} + \varepsilon}$ имеет те же единицы, что и параметр — шаг автоматически согласован по масштабу.

Adadelta [3]

Развитие RMSProp: вообще **не требует** глобального шага α. Вместо него масштабирует шаг через накопленные обновления параметров:

$$v_j^{(k)} = \gamma v_j^{(k-1)} + (1-\gamma)\left(g_j^{(k)}\right)^2$$

$$\tilde{g}_j^{(k)} = \frac{\sqrt{\Delta x_j^{(k-1)} + \varepsilon}}{\sqrt{v_j^{(k)} + \varepsilon}}\, g_j^{(k)}$$

$$x_j^{(k)} = x_j^{(k-1)} - \tilde{g}_j^{(k)}, \qquad \Delta x_j^{(k)} = \rho\, \Delta x_j^{(k-1)} + (1-\rho)\left(\tilde{g}_j^{(k)}\right)^2$$

Замечания:

- Числитель $\sqrt{\Delta x_j^{(k-1)} + \varepsilon}$ имеет те же единицы, что и параметр — шаг автоматически согласован по масштабу.
- Полезен, когда масштабы параметров между слоями существенно различаются.

[3] Zeiler (2012). ADADELTA: An Adaptive Learning Rate Method.

Adam [4] [5]

Комбинирует элементы AdaGrad и RMSProp. Экспоненциально затухающее среднее градиентов и квадратов градиентов:

EMA:

$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$

$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Коррекция:

$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$

$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}$$

Замечания:

- Корректирует смещение моментов к нулю на старте.

Adam [4] [5]

Комбинирует элементы AdaGrad и RMSProp. Экспоненциально затухающее среднее градиентов и квадратов градиентов:

EMA:
$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$
$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Коррекция:
$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$
$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}$$

Замечания:

- Корректирует смещение моментов к нулю на старте.
- Одна из самых цитируемых работ в области ML.

Adam [4] [5]

Комбинирует элементы AdaGrad и RMSProp. Экспоненциально затухающее среднее градиентов и квадратов градиентов:

EMA:
$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$
$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2) \left(g_j^{(k)}\right)^2$$

Коррекция:
$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$
$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}$$

Замечания:

- Корректирует смещение моментов к нулю на старте.
- Одна из самых цитируемых работ в области ML.
- Не сходится для некоторых простых задач (даже выпуклых).

Adam [4] [5]

$$\min_w \; Loss(w) + \lambda \frac{1}{2}\|w\|_2^2$$

$$\nabla loss + \lambda w$$

Комбинирует элементы AdaGrad и RMSProp. Экспоненциально затухающее среднее градиентов и квадратов градиентов:

EMA:
$$m_j^{(k)} = \beta_1 m_j^{(k-1)} + (1-\beta_1) g_j^{(k)}$$
$$v_j^{(k)} = \beta_2 v_j^{(k-1)} + (1-\beta_2)\left(g_j^{(k)}\right)^2$$

Коррекция:
$$\hat{m}_j = \frac{m_j^{(k)}}{1-\beta_1^k}$$
$$\hat{v}_j = \frac{v_j^{(k)}}{1-\beta_2^k}$$

Обновление:
$$x_j^{(k)} = x_j^{(k-1)} - \alpha \frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon}$$

Замечания:

- Корректирует смещение моментов к нулю на старте.
- Одна из самых цитируемых работ в области ML.
- Не сходится для некоторых простых задач (даже выпуклых).
- Гораздо лучше работает для языковых моделей, чем для задач компьютерного зрения — открытый вопрос «почему».

Adam ~ SGD + Momentum

[4] Kingma, Ba (2014). Adam: A Method for Stochastic Optimization.
[5] Reddi, Kale, Kumar (2018). On the Convergence of Adam and Beyond.

AdamW [6]

Решает проблему ℓ_2-регуляризации в адаптивных оптимизаторах. При стандартном подходе ℓ_2-регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, порождая градиентный член λx. В Adam этот член делится на $(\sqrt{\hat{v}_j} + \varepsilon)$ — сила регуляризации оказывается зависимой от магнитуд градиентов, что нежелательно.

AdamW **разделяет** weight decay и адаптацию градиентов:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon} + \lambda x_j^{(k-1)} \right)$$

Замечания:

- Член weight decay $\lambda x_j^{(k-1)}$ добавляется *после* адаптивного градиентного шага и не масштабируется на $\sqrt{\hat{v}_j}$.

AdamW [6]

Решает проблему ℓ_2-регуляризации в адаптивных оптимизаторах. При стандартном подходе ℓ_2-регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, порождая градиентный член λx. В Adam этот член делится на $(\sqrt{\hat{v}_j} + \varepsilon)$ — сила регуляризации оказывается зависимой от магнитуд градиентов, что нежелательно.

AdamW **разделяет** weight decay и адаптацию градиентов:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \varepsilon} + \lambda x_j^{(k-1)} \right)$$

Замечания:

- Член weight decay $\lambda x_j^{(k-1)}$ добавляется *после* адаптивного градиентного шага и не масштабируется на $\sqrt{\hat{v}_j}$.
- Широко используется при обучении трансформеров и больших моделей; выбор по умолчанию в HuggingFace Trainer.

[6] Loshchilov, Hutter (2017). Decoupled Weight Decay Regularization.

Adam не сходится: постановка [7]

Рассмотрим **простейшую** возможную задачу — выпуклую, гладкую, сильно выпуклую, с единственным глобальным минимумом в нуле:

$$L(w) = w^2, \qquad \nabla L(w) = 2w, \qquad w \in \mathbb{R}.$$

Обновление Adam (одномерный случай, без коррекции смещения и для упрощения $\varepsilon = 0$):

$$\begin{aligned} g_t &= 2w_{t-1}, \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\ w_t &= w_{t-1} - \eta \frac{m_t}{\sqrt{v_t}}. \end{aligned}$$

[7] Reddi et al. (2018). On the Convergence of Adam and Beyond.

Adam не сходится: постановка [7]

Рассмотрим **простейшую** возможную задачу — выпуклую, гладкую, сильно выпуклую, с единственным глобальным минимумом в нуле:

$$L(w) = w^2, \qquad \nabla L(w) = 2w, \qquad w \in \mathbb{R}.$$

Обновление Adam (одномерный случай, без коррекции смещения и для упрощения $\varepsilon = 0$):

$$\begin{aligned} g_t &= 2w_{t-1}, \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \\ w_t &= w_{t-1} - \eta \frac{m_t}{\sqrt{v_t}}. \end{aligned}$$

Чего мы **ждём** от корректного метода: $w_t \to 0$ и длина шага $|w_t - w_{t-1}| \to 0$ по мере приближения к оптимуму. У GD с $\eta < 1/2$ это так: $w_t = (1-2\eta) w_{t-1}$, экспоненциальная сходимость.

Покажем, что у Adam **этого не происходит**.

[7] Reddi et al. (2018). On the Convergence of Adam and Beyond.

Adam не сходится: анализ масштаба при малых $|w|$

Предположим, что итерации находятся в окрестности нуля и величина $|w_t|$ ведёт себя как масштаб $|w| \to 0$.
Что происходит с m_t и v_t?

Adam не сходится: анализ масштаба при малых $|w|$

Предположим, что итерации находятся в окрестности нуля и величина $|w_t|$ ведёт себя как масштаб $|w| \to 0$. Что происходит с m_t и v_t?

m_t **— линеен по** w**.** Это сумма (взвешенная) прошлых градиентов $g_\tau = 2w_{\tau-1}$, каждый из которых линеен по w:

$$m_t \;=\; (1-\beta_1)\sum_{\tau\le t}\beta_1^{t-\tau}\,2w_{\tau-1} \;\propto\; w.$$

Adam не сходится: анализ масштаба при малых $|w|$

Предположим, что итерации находятся в окрестности нуля и величина $|w_t|$ ведёт себя как масштаб $|w| \to 0$. Что происходит с m_t и v_t?

m_t — **линеен по** w**.** Это сумма (взвешенная) прошлых градиентов $g_\tau = 2w_{\tau-1}$, каждый из которых линеен по w:

$$m_t \;=\; (1-\beta_1)\sum_{\tau\le t}\beta_1^{t-\tau}\,2w_{\tau-1} \;\propto\; w.$$

v_t — **квадратичен по** w**.** Сумма прошлых $g_\tau^2 = 4w_{\tau-1}^2$, каждый член квадратичен:

$$v_t \;=\; (1-\beta_2)\sum_{\tau\le t}\beta_2^{t-\tau}\,4w_{\tau-1}^2 \;\propto\; w^2.$$

Adam не сходится: анализ масштаба при малых $|w|$

Предположим, что итерации находятся в окрестности нуля и величина $|w_t|$ ведёт себя как масштаб $|w| \to 0$. Что происходит с m_t и v_t?

m_t — **линеен по** w**.** Это сумма (взвешенная) прошлых градиентов $g_\tau = 2w_{\tau-1}$, каждый из которых линеен по w:

$$m_t \;=\; (1-\beta_1)\sum_{\tau\le t}\beta_1^{t-\tau}\,2w_{\tau-1} \;\propto\; w.$$

v_t — **квадратичен по** w**.** Сумма прошлых $g_\tau^2 = 4w_{\tau-1}^2$, каждый член квадратичен:

$$v_t \;=\; (1-\beta_2)\sum_{\tau\le t}\beta_2^{t-\tau}\,4w_{\tau-1}^2 \;\propto\; w^2.$$

Следовательно:

$$\sqrt{v_t} \;\propto\; |w| \quad\Longrightarrow\quad \frac{m_t}{\sqrt{v_t}} \;\approx\; \operatorname{sign}(w) \;=\; \pm 1$$

Это отношение **не зависит** от $|w|$: какой бы маленькой ни стала итерация, нормированный градиент остаётся порядка единицы.

Adam не сходится: предельный цикл

Из предыдущего шага: шаг Adam в окрестности нуля имеет постоянную **по модулю** длину

$$|w_t - w_{t-1}| \;=\; \eta \left| \frac{m_t}{\sqrt{v_t}} \right| \;\approx\; \eta,$$

независимо от того, насколько w близко к оптимуму.

Adam не сходится: предельный цикл

Из предыдущего шага: шаг Adam в окрестности нуля имеет постоянную **по модулю** длину

$$|w_t - w_{t-1}| \;=\; \eta \left| \frac{m_t}{\sqrt{v_t}} \right| \;\approx\; \eta,$$

независимо от того, насколько w близко к оптимуму.

Что это означает. Если $|w_{t-1}| < \eta/2$, то шаг $\pm\eta$ «перепрыгивает» через ноль и оказывается с противоположной стороны на расстоянии $\approx \eta/2$. На следующем шаге — обратно.

Adam не сходится: предельный цикл

Из предыдущего шага: шаг Adam в окрестности нуля имеет постоянную **по модулю** длину

$$|w_t - w_{t-1}| \;=\; \eta \left| \frac{m_t}{\sqrt{v_t}} \right| \;\approx\; \eta,$$

независимо от того, насколько w близко к оптимуму.

Что это означает. Если $|w_{t-1}| < \eta/2$, то шаг $\pm\eta$ «перепрыгивает» через ноль и оказывается с противоположной стороны на расстоянии $\approx \eta/2$. На следующем шаге — обратно.

Получаем **предельный цикл**:

$$w_t \;\in\; \{-\eta/2,\; +\eta/2\}, \qquad t \to \infty.$$

Метод **не сходится** ни к точке, ни даже к нулю по среднему.

Adam не сходится: предельный цикл

Из предыдущего шага: шаг Adam в окрестности нуля имеет постоянную **по модулю** длину

$$|w_t - w_{t-1}| \;=\; \eta \left| \frac{m_t}{\sqrt{v_t}} \right| \;\approx\; \eta,$$

независимо от того, насколько w близко к оптимуму.

Что это означает. Если $|w_{t-1}| < \eta/2$, то шаг $\pm\eta$ «перепрыгивает» через ноль и оказывается с противоположной стороны на расстоянии $\approx \eta/2$. На следующем шаге — обратно.

Получаем **предельный цикл**:

$$w_t \;\in\; \{-\eta/2,\; +\eta/2\}, \qquad t \to \infty.$$

Метод **не сходится** ни к точке, ни даже к нулю по среднему.

Сравнение с SGD. Для $L(w) = w^2$ обновление SGD:

$$w_t = w_{t-1} - \eta \cdot 2w_{t-1} = (1 - 2\eta)w_{t-1}.$$

Шаг **пропорционален** w, обнуляется в пределе $\Rightarrow$ сходимость.

Adam не сходится: иллюстрация

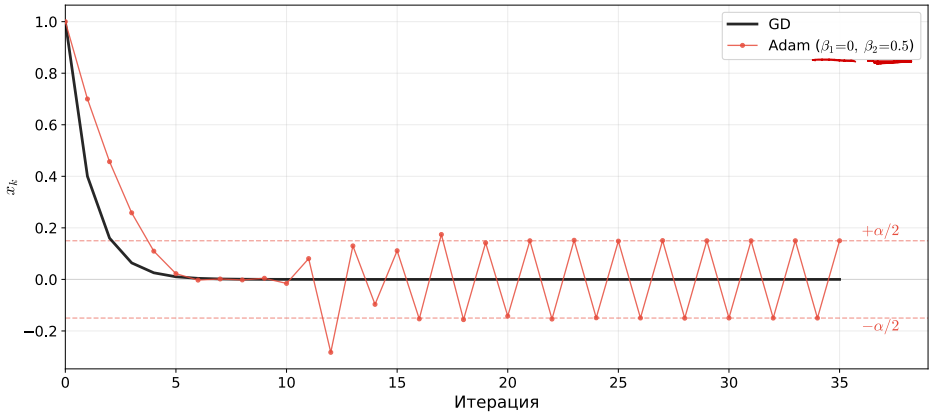
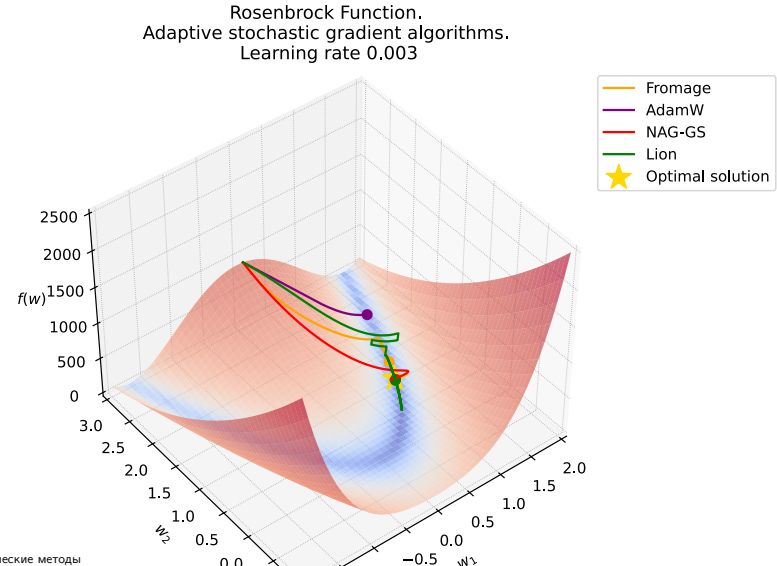


Figure 1: GD сходится к нулю, Adam выходит на предельный цикл амплитуды $\sim \alpha/2$.

Много оптимизаторов — как сравнивать?



AlgoPerf benchmark [8]

- **AlgoPerf** — стандартизированный бенчмарк для сравнения алгоритмов обучения нейросетей по двум регламентам:

AlgoPerf benchmark [8]

- **AlgoPerf** — стандартизированный бенчмарк для сравнения алгоритмов обучения нейросетей по двум регламентам:
  - **External Tuning** — подбор гиперпараметров с ограниченными ресурсами (5 попыток);

AlgoPerf benchmark [8]

- **AlgoPerf** — стандартизированный бенчмарк для сравнения алгоритмов обучения нейросетей по двум регламентам:
  - **External Tuning** — подбор гиперпараметров с ограниченными ресурсами (5 попыток);
  - **Self-Tuning** — автоматическая настройка на одной машине ($3\times$ бюджет).

AlgoPerf benchmark [8]

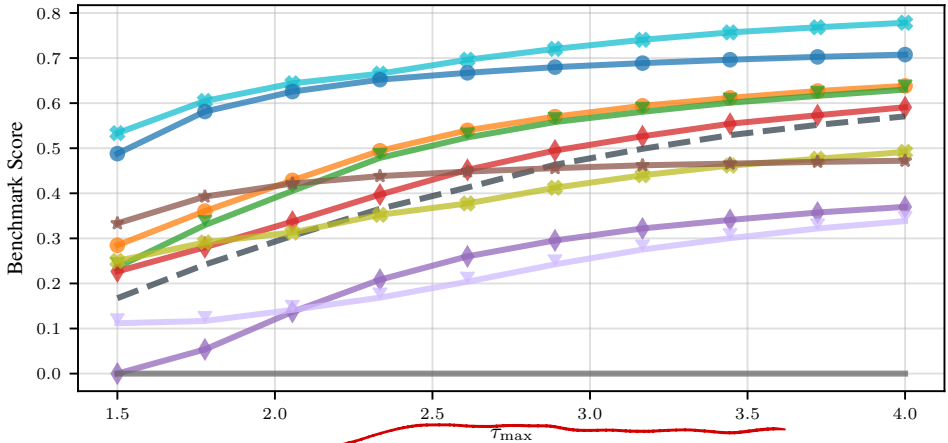
- **AlgoPerf** — стандартизированный бенчмарк для сравнения алгоритмов обучения нейросетей по двум регламентам:
  - **External Tuning** — подбор гиперпараметров с ограниченными ресурсами (5 попыток);
  - **Self-Tuning** — автоматическая настройка на одной машине ($3\times$ бюджет).
- **Оценка** агрегируется через профили производительности; финальный балл — нормализованная площадь под профилем (1.0 = быстрейший на всех задачах).

AlgoPerf benchmark [8]

- **AlgoPerf** — стандартизированный бенчмарк для сравнения алгоритмов обучения нейросетей по двум регламентам:
  - **External Tuning** — подбор гиперпараметров с ограниченными ресурсами (5 попыток);
  - **Self-Tuning** — автоматическая настройка на одной машине ($3\times$ бюджет).
- **Оценка** агрегируется через профили производительности; финальный балл — нормализованная площадь под профилем (1.0 = быстрейший на всех задачах).
- **Стоимость** оценки: $\sim 49\,240$ часов работы на 8 NVIDIA V100 GPU.

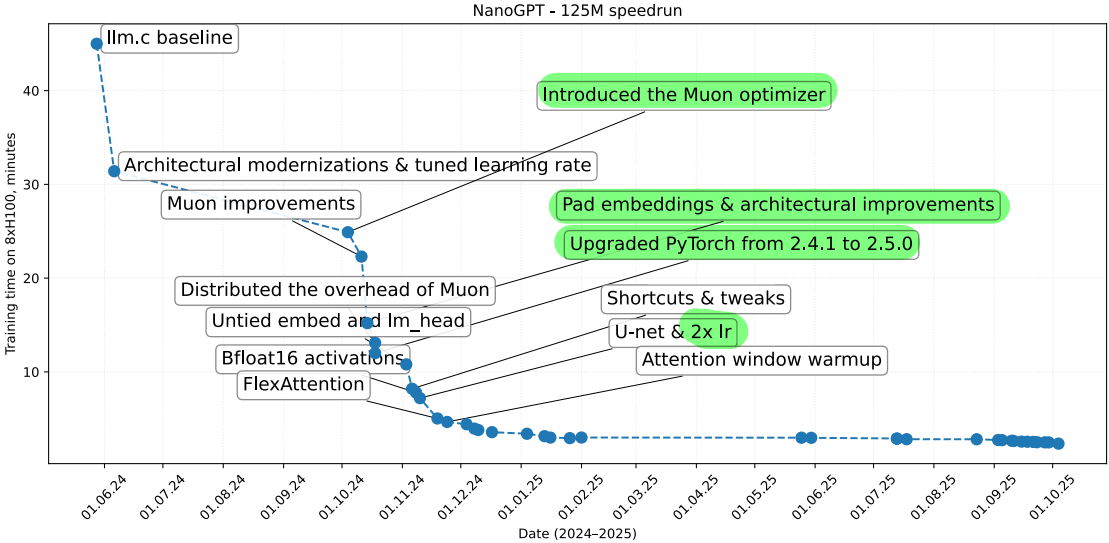
[8] Dahl et al. (2023). Benchmarking Neural Network Training Algorithms.

AlgoPerf benchmark: результаты



Мотивация: сравнение оптимизаторов

NanoGPT speedrun [9]



[9] Источник

Работают ли трюки, если увеличить размер модели?

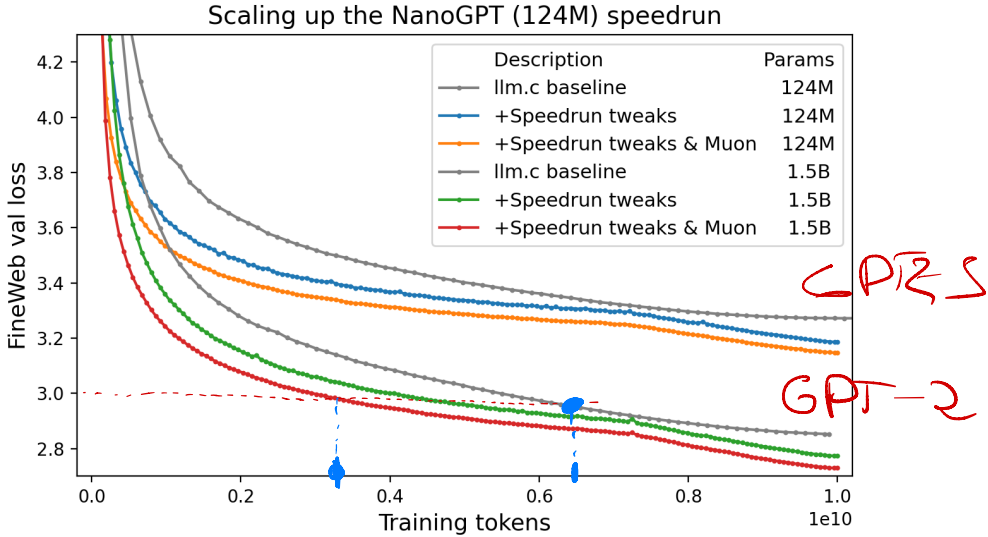


Figure 2: Источник

Сравнение оптимизаторов на NanoGPT speedrun

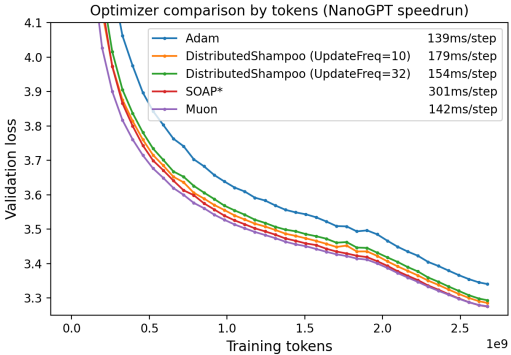


Figure 3: По числу токенов

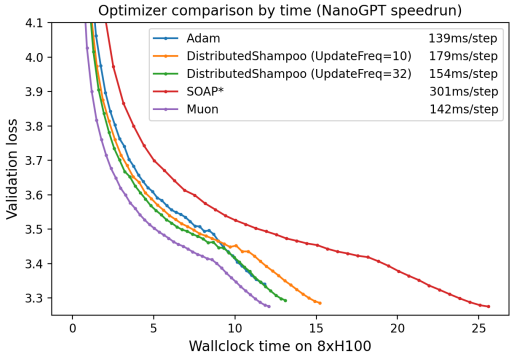
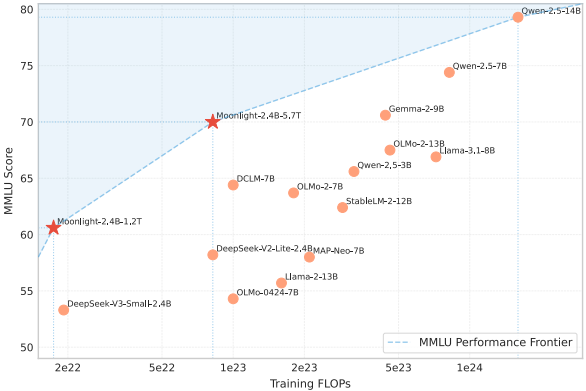


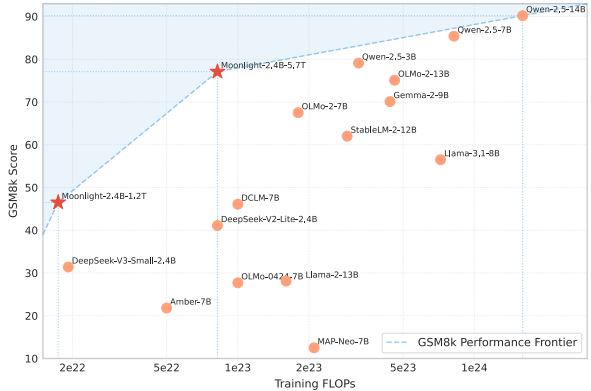
Figure 4: По wallclock time (8×H100)

Muon (фиолетовый) — лучший по обоим метрикам: быстрейшая сходимость по токенам и самый дешёвый по времени (142 ms/step).

Shampoo и Muon

Новый подход к оптимизации [10]





Модели, отмеченные звёздочкой, были обучены методом Muon, остальные модели были обучены другими алгоритмами оптимизации.

[10] Kimi K2: Open Agentic Intelligence

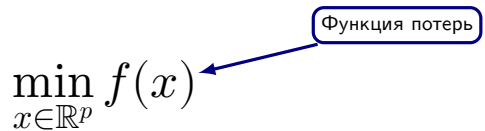
Интуиция за методом Muon [11]

$$\min_{x \in \mathbb{R}^p} f(x)$$

$$f(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \mathcal{O}(\|x - x_k\|_2^2).$$

[11] Презентация R. Gower

Интуиция за методом Muon [11]



$$f(x) = f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \mathcal{O}(\|x - x_k\|_2^2).$$

[11] Презентация R. Gower

Интуиция за методом Muon [11]

Функция потерь

$$\min_{x \in \mathbb{R}^p} f(x)$$

$$f(x) = \underbrace{f(x_k) + \langle \nabla f(x_k), x - x_k \rangle}_{\text{Линейная аппроксимация}} + \mathcal{O}(\|x - x_k\|_2^2).$$

Интуиция за методом Muon [11]

Функция потерь

$$\min_{x \in \mathbb{R}^p} f(x)$$

$$f(x) = \underbrace{f(x_k) + \langle \nabla f(x_k), x - x_k \rangle}_{\text{Линейная аппроксимация}} + \mathcal{O}(\|x - x_k\|_2^2).$$

Хорошее приближение в окрестности x_k

[11] Презентация R. Gower

Интуиция за методом Muon. Градиентный спуск

$$\nabla f(x_k) + \frac{1}{\alpha}(x_{k+1} - x_k) = 0$$

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right)$$

Интуиция за методом Muon. Градиентный спуск

$$
\begin{aligned}
x_{k+1} &= \operatorname*{argmin}_{x \in \mathbb{R}^p} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right) \\
&= x_k - \alpha \nabla f(x_k)
\end{aligned}
$$

Интуиция за методом Muon. Градиентный спуск

Штраф за дальность от x_k

$$x_{k+1} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right)$$
$$= x_k - \alpha \nabla f(x_k)$$

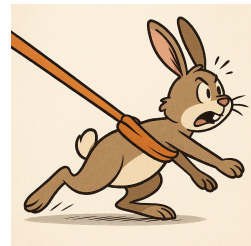
Интуиция за методом Muon. Градиентный спуск

Штраф за дальность от x_k

$$x_{k+1} = \underset{x \in \mathbb{R}^p}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right)$$

$$= x_k - \alpha \nabla f(x_k)$$

Шаг обучения / коэффициент регуляризации



Интуиция за методом Muon. Нормированный градиентный спуск

$$x_{k+1} = \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle\right)$$

Интуиция за методом Muon. Нормированный градиентный спуск

$$
\begin{aligned}
x_{k+1} &= \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle \right) \\
&= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}
\end{aligned}
$$

Интуиция за методом Muon. Нормированный градиентный спуск

$$x_{k+1} = \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle\right)$$

$$= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}$$

Ограничение на длину шага

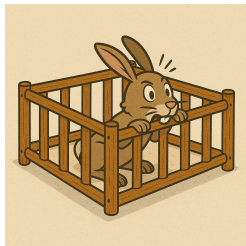
Интуиция за методом Muon. Нормированный градиентный спуск

$$x_{k+1} = \underset{\|x - x_k\|_2 = \alpha}{\operatorname{argmin}} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle\right)$$

$$= x_k - \alpha \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|_2}$$

Ограничение на длину шага

Параметр ограничения / шаг обучения



Что насчёт других норм?

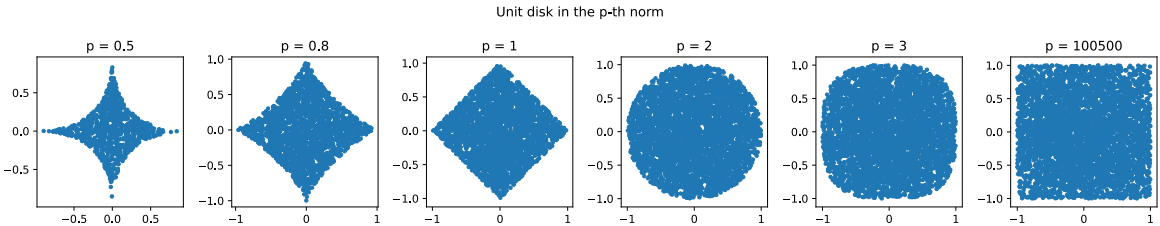


Figure 5: Примеры шаров в разных нормах

Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

$$(\|g\|_p)^* = \|g\|_q$$

$$\frac{1}{p} + \frac{1}{q} = 1$$

- Linear Minimization Oracle:

$$\text{LMO}_{\|\cdot\|}(g) = \operatorname*{argmin}_{\|x\|=1} \langle g, x \rangle$$

Для неевклидовых норм нужно ввести несколько определений

- Сопряжённая норма:

$$\|g\|^* = \sup_{\|x\|=1} \langle g, x \rangle$$

- Linear Minimization Oracle:

$$\mathsf{LMO}_{\|\cdot\|}(g) = \operatorname*{argmin}_{\|x\|=1} \langle g, x \rangle$$

- Важное свойство, связывающее эти два понятия:

$$\langle g, \mathsf{LMO}_{\|\cdot\|}(g) \rangle = -\|g\|^*$$

Неевклидовы записи методов [12]

Неевклидов градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left(f(x_k) + \langle g, x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|^2 \right) = x_k + \alpha \|g\|^* \mathsf{LMO}_{\|\cdot\|}(g)$$

$x_k - \alpha \nabla f(x_k)$

$$x_{k+1} = \operatorname*{argmin}_{\|x - x_k\| = \alpha} \left(f(x_k) + \langle g, x - x_k \rangle \right)$$

[12] Old Optimizer, New Norm: An Anthology

Неевклидовы записи методов [12]

Неевклидов градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$x_{k+1} = \operatorname*{argmin}_{x \in \mathbb{R}^p} \left(f(x_k) + \langle g, x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|^2 \right) = x_k + \alpha \|g\|^* \mathsf{LMO}_{\|\cdot\|}(g)$$

Неевклидов нормированный градиентный спуск

Для вектора градиента $g = \nabla f(x_k)$ и шага $\alpha > 0$:

$$x_{k+1} = \operatorname*{argmin}_{\|x - x_k\| = \alpha} \left(f(x_k) + \langle g, x - x_k \rangle \right) = x_k + \alpha \mathsf{LMO}_{\|\cdot\|}(g)$$

$$-\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

[12] Old Optimizer, New Norm: An Anthology

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \qquad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \qquad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{op}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \qquad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{op}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.
- Вся логика переносится: вместо вектора ищем «лучшее направление спуска» среди матриц заданной нормы.

В нейросетях параметры — матрицы

- В линейных слоях, attention, embedding-слоях параметр — матрица весов

$$W \in \mathbb{R}^{d \times n}, \qquad G_k = \nabla_W f(W_k) \in \mathbb{R}^{d \times n}.$$

- Естественно использовать **матричные нормы**: операторную $\|\cdot\|_{\text{op}}$, ядерную $\|\cdot\|_{\text{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.
- Вся логика переносится: вместо вектора ищем «лучшее направление спуска» среди матриц заданной нормы.
- Скалярное произведение:

$$\langle A, B \rangle := \operatorname{tr}(A^\top B) = \sum_{ij} A_{ij} B_{ij}.$$

Неевклидов нормированный спуск для матриц

Пусть заданы матричная норма $\|\cdot\|$ и шаг $\lambda > 0$. Тогда нормированный шаг по матрице W:

$$W_{k+1} = \underset{\|W - W_k\| = \lambda}{\operatorname{argmin}} \Big(f(W_k) + \langle G_k, W - W_k \rangle \Big)$$
$$= W_k + \lambda \mathsf{LMO}_{\|\cdot\|}(G_k),$$

где

$$\mathsf{LMO}_{\|\cdot\|}(G) = \underset{\|W\| = 1}{\operatorname{argmin}} \langle G, W \rangle$$

— тот же самый LMO, только теперь он ищет **матрицу** единичной нормы, дающую наибольшее убывание линейного приближения.

$W_{k+1} = W_k - \lambda UV^T$

Muon

Операторная норма и быстрый расчёт ($UV^\top$)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{op}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO по операторной норме:

$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$

то есть оптимальное направление — **полярный фактор** матрицы G_k (ортогональная часть полярного разложения $G = UP$).

Операторная норма и быстрый расчёт ($UV^\top$)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{op}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO по операторной норме:
$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$
то есть оптимальное направление — **полярный фактор** матрицы G_k (ортогональная часть полярного разложения $G = UP$).
- Проблема: полное SVD стоит $O(d^3)$ на каждом шаге. Но нам нужен только произведение $UV^\top$, а не сами U, Σ, V по отдельности.

Операторная норма и быстрый расчёт ($UV^\top$)

Рассмотрим операторную (спектральную) норму $\|\cdot\|_{\text{op}}$. Пусть

$$G_k = U\Sigma V^\top$$

— редуцированное SVD градиента. Тогда

- LMO по операторной норме:
$$\text{LMO}_{\|\cdot\|}(G) = -UV^\top,$$
то есть оптимальное направление — **полярный фактор** матрицы G_k (ортогональная часть полярного разложения $G = UP$).
- Проблема: полное SVD стоит $O(d^3)$ на каждом шаге. Но нам нужен только произведение $UV^\top$, а не сами U, Σ, V по отдельности.
- Итерации **Newton–Schulz** используют только матричные умножения и дают приближение $UV^\top$ за 5–10 шагов — снимают вычислительное узкое место Muon.

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.

Замечания:

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.

Замечания:

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобусловители $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$ (обратный матричный корень).

Замечания:

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобусловители $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$ (обратный матричный корень).
4. Обновление: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Замечания:

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **Of** deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобусловители $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$ (обратный матричный корень).
4. Обновление: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Замечания:

- Учитывает информацию о кривизне эффективнее методов первого порядка.

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобусловители $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$ (обратный матричный корень).
4. Обновление: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Замечания:

- Учитывает информацию о кривизне эффективнее методов первого порядка.
- Вычислительно дороже Adam, но может сходиться быстрее по числу шагов.

Shampoo [13]

Shampoo = **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks — метод, вдохновлённый оптимизацией второго порядка.

Основная идея: Приближает полноматричный предобусловитель AdaGrad через произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$:

1. Вычислить градиент G_k.
2. Обновить статистику $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобусловители $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$ (обратный матричный корень).
4. Обновление: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Замечания:

- Учитывает информацию о кривизне эффективнее методов первого порядка.
- Вычислительно дороже Adam, но может сходиться быстрее по числу шагов.
- Требует аккуратной реализации (эффективное вычисление обратных матричных корней).

[13] Gupta, Koren, Singer (2018). Shampoo: Preconditioned Stochastic Tensor Optimization. ICML.

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \leq 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \leq 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

- **SGD** (норма Фробениуса): $D^* = -G/\|G\|_F$ — направление $\propto$ градиенту

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \leq 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

- **SGD** (норма Фробениуса): $D^* = -G/\|G\|_F$ — направление $\propto$ градиенту
- **SignGD** (ℓ_∞-норма): $D^* = -\operatorname{sign}(G)$ — знак каждого элемента

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \le 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

- **SGD** (норма Фробениуса): $D^* = -G/\|G\|_F$ — направление $\propto$ градиенту
- **SignGD** (ℓ_∞-норма): $D^* = -\operatorname{sign}(G)$ — знак каждого элемента
- **Muon** (спектральная норма): $D^* = -UV^\top$ — полярный фактор градиента

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \leq 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

- **SGD** (норма Фробениуса): $D^* = -G/\|G\|_F$ — направление $\propto$ градиенту
- **SignGD** (ℓ_∞-норма): $D^* = -\operatorname{sign}(G)$ — знак каждого элемента
- **Muon** (спектральная норма): $D^* = -UV^\top$ — полярный фактор градиента

[14] J. Bernstein "Deriving Muon". 2025.
[15] Kovalev, D. (2025). Understanding Gradient Orthogonalization.

Muon: вывод через наискорейший спуск в спектральной норме [14] [15]

Задача наискорейшего спуска для матричного параметра W:

$$D^* = \arg\min_{\|D\|_\sigma \le 1} \operatorname{tr}\left[\nabla L(W)^\top D\right]$$

где $\|\cdot\|_\sigma$ — спектральная норма (максимальное сингулярное число).

- **SGD** (норма Фробениуса): $D^* = -G/\|G\|_F$ — направление $\propto$ градиенту
- **SignGD** (ℓ_∞-норма): $D^* = -\operatorname{sign}(G)$ — знак каждого элемента
- **Muon** (спектральная норма): $D^* = -UV^\top$ — полярный фактор градиента

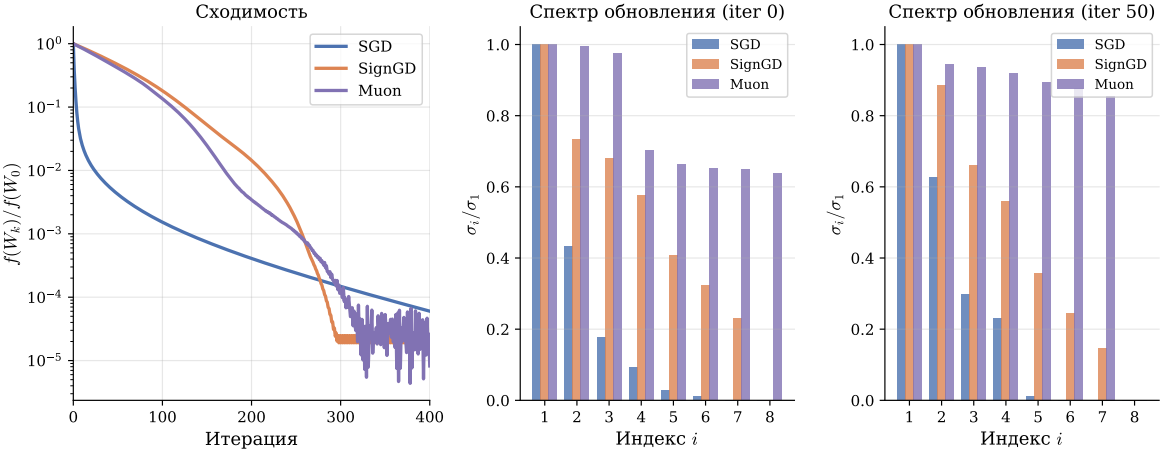
Двойственная норма к спектральной — **ядерная** (сумма сингулярных чисел):

$$\|G\|_* = \sum_i \sigma_i(G), \qquad \arg\min_{\|D\|_\sigma \le 1} \langle G, D\rangle = -UV^\top$$

[14] J. Bernstein "Deriving Muon". 2025.
[15] Kovalev, D. (2025). Understanding Gradient Orthogonalization.

Три нормы — три оптимизатора: эксперимент

Наискорейший спуск в трёх нормах: $\|\cdot\|_F$ (SGD), $\|\cdot\|_\infty$ (SignGD), $\|\cdot\|_\sigma$ (Muon)



$f(W) = \frac{1}{2}\|\Sigma W\|_F^2$, $W \in \mathbb{R}^{8\times 8}$, $\kappa(\Sigma) = 20$. **Слева**: сходимость трёх методов. **Справа**: спектр обновления ΔW. SGD концентрирует энергию на 1–2 сингулярных направлениях, Muon — распределяет равномерно.

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

$$= W_t - \eta (U S^2 U^\top)^{-1/4} (U S V^\top) (V S^2 V^\top)^{-1/4}$$

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

$$= W_t - \eta (U S^2 U^\top)^{-1/4} (U S V^\top) (V S^2 V^\top)^{-1/4}$$

$$= W_t - \eta (U S^{-1/2} U^\top)(U S V^\top)(V S^{-1/2} V^\top)$$

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

$$= W_t - \eta (U S^2 U^\top)^{-1/4} (U S V^\top) (V S^2 V^\top)^{-1/4}$$

$$= W_t - \eta (U S^{-1/2} U^\top)(U S V^\top)(V S^{-1/2} V^\top)$$

$$= W_t - \eta U S^{-1/2} S S^{-1/2} V^\top$$

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

$$= W_t - \eta (U S^2 U^\top)^{-1/4} (U S V^\top) (V S^2 V^\top)^{-1/4}$$

$$= W_t - \eta (U S^{-1/2} U^\top)(U S V^\top)(V S^{-1/2} V^\top)$$

$$= W_t - \eta U S^{-1/2} S S^{-1/2} V^\top$$

$$= W_t - \eta U V^\top$$

[16] J. Bernstein "Deriving Muon"

Muon: упрощение Shampoo [16]

$$W_{t+1} = W_t - \eta \underbrace{(G_t G_t^\top)^{-1/4} G_t (G_t^\top G_t)^{-1/4}}_{\text{Shampoo}}$$

$$= W_t - \eta (U S^2 U^\top)^{-1/4} (U S V^\top) (V S^2 V^\top)^{-1/4}$$

$$= W_t - \eta (U S^{-1/2} U^\top)(U S V^\top)(V S^{-1/2} V^\top)$$

$$= W_t - \eta U S^{-1/2} S S^{-1/2} V^\top$$

$$= W_t - \eta U V^\top$$

Ключевой инсайт: Shampoo с «идеальной» статистикой (один градиент вместо накопленных) — это просто ортогонализация градиента. Все сингулярные числа обновления равны 1.

[16] J. Bernstein "Deriving Muon"

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - ax_k).$$

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - a x_k).$$

Невязка $u_k = 1 - a x_k$ удовлетворяет $u_{k+1} = u_k^2$ — **квадратичная сходимость** при $|u_0| < 1$.

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - ax_k).$$

Невязка $u_k = 1 - ax_k$ удовлетворяет $u_{k+1} = u_k^2$ — **квадратичная сходимость** при $|u_0| < 1$.

Заменим скаляры на матрицы: $x_k \to X_k$, $a \to A$:

$$\boxed{X_{k+1} = X_k(2I - AX_k) \;\longrightarrow\; A^{-1}}$$

Ни одного обращения матрицы — только умножения.

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - ax_k).$$

Невязка $u_k = 1 - ax_k$ удовлетворяет $u_{k+1} = u_k^2$ — **квадратичная сходимость** при $|u_0| < 1$.

Заменим скаляры на матрицы: $x_k \to X_k$, $a \to A$:

$$\boxed{X_{k+1} = X_k(2I - AX_k) \;\longrightarrow\; A^{-1}}$$

Ни одного обращения матрицы — только умножения.

Теперь **другая задача**: вычислить $1/\sqrt{a}$. Метод Ньютона к $f(x) = 1/x^2 - a = 0$:

$$x_{k+1} = x_k - \frac{1/x_k^2 - a}{-2/x_k^3} = \tfrac{1}{2}x_k(3 - ax_k^2).$$

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - ax_k).$$

Невязка $u_k = 1 - ax_k$ удовлетворяет $u_{k+1} = u_k^2$ — **квадратичная сходимость** при $|u_0| < 1$.

Заменим скаляры на матрицы: $x_k \to X_k$, $a \to A$:

$$\boxed{X_{k+1} = X_k(2I - AX_k) \;\longrightarrow\; A^{-1}}$$

Ни одного обращения матрицы — только умножения.

Теперь **другая задача**: вычислить $1/\sqrt{a}$. Метод Ньютона к $f(x) = 1/x^2 - a = 0$:

$$x_{k+1} = x_k - \frac{1/x_k^2 - a}{-2/x_k^3} = \tfrac{1}{2}x_k(3 - ax_k^2).$$

Полярный фактор: $UV^\top = G(G^\top G)^{-1/2}$, т.е. нужен **обратный матричный корень** от $G^\top G$.

$G = U\Sigma V^\top$

Откуда берётся итерация Newton–Schulz

Задача: вычислить $1/a$ без деления — только сложениями и умножениями.

Применим метод Ньютона к $f(x) = 1/x - a = 0$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k(2 - ax_k).$$

Невязка $u_k = 1 - ax_k$ удовлетворяет $u_{k+1} = u_k^2$ — **квадратичная сходимость** при $|u_0| < 1$.

Заменим скаляры на матрицы: $x_k \to X_k$, $a \to A$:

$$\boxed{X_{k+1} = X_k(2I - AX_k) \;\longrightarrow\; A^{-1}}$$

Ни одного обращения матрицы — только умножения.

Теперь **другая задача**: вычислить $1/\sqrt{a}$. Метод Ньютона к $f(x) = 1/x^2 - a = 0$:

$$x_{k+1} = x_k - \frac{1/x_k^2 - a}{-2/x_k^3} = \tfrac{1}{2}x_k(3 - ax_k^2).$$

Полярный фактор: $UV^\top = G(G^\top G)^{-1/2}$, т.е. нужен **обратный матричный корень** от $G^\top G$.

Заменим $a \to X_k^\top X_k$ в скалярной формуле для $1/\sqrt{a}$:

$$\boxed{X_{k+1} = \tfrac{1}{2}X_k(3I - X_k^\top X_k), \qquad \sigma_i(X_k) \to 1}$$

POLAR EXPRESS CANS

Newton-Schultz

Muon: алгоритм и Newton-Schulz

Algorithm 1 Proposed optimizer

Require: Learning rate η, momentum μ, weight decay λ

1: Initialize $B_0 \leftarrow 0$
2: **for** $t = 1, \dots$ **do**
3: Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
4: $B_t \leftarrow \mu B_{t-1} + G_t$
5: $G_t \leftarrow \mu B_t + G_t$
6: $O_t \leftarrow \mathrm{NewtonSchulz5}(G_t)$
7: Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta(O_t + \lambda\theta_{t-1})$
8: **end for**
9: **return** θ_t

Newton-Schulz вместо SVD для вычисления $UV^\top$:

$$X_0 = G/\|G\|_F$$

$$X_{k+1} = aX_k + bX_kX_k^\top X_k + c(X_kX_k^\top)^2X_k$$

с коэффициентами a=3.4445, b=−4.7750, c=2.0315.

- **5 итераций** достаточно на практике

Muon: алгоритм и Newton-Schulz

Algorithm 1 Proposed optimizer

Require: Learning rate η, momentum μ, weight decay λ

1: Initialize $B_0 \leftarrow 0$
2: **for** $t = 1, \ldots$ **do**
3: Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
4: $B_t \leftarrow \mu B_{t-1} + G_t$
5: $G_t \leftarrow \mu B_t + G_t$
6: $O_t \leftarrow \mathrm{NewtonSchulz5}(G_t)$
7: Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta(O_t + \lambda\theta_{t-1})$
8: **end for**
9: **return** θ_t

Newton-Schulz вместо SVD для вычисления $UV^\top$:

$$X_0 = G/\|G\|_F$$

$$X_{k+1} = aX_k + bX_kX_k^\top X_k + c(X_kX_k^\top)^2X_k$$

с коэффициентами a=3.4445, b=−4.7750, c=2.0315.

- **5 итераций** достаточно на практике
- Только матричные умножения — эффективно на GPU

Muon: алгоритм и Newton-Schulz

Algorithm 1 Proposed optimizer

Require: Learning rate η, momentum μ, weight decay λ

1: Initialize $B_0 \leftarrow 0$
2: **for** $t = 1, \ldots$ **do**
3: Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
4: $B_t \leftarrow \mu B_{t-1} + G_t$
5: $G_t \leftarrow \mu B_t + G_t$
6: $O_t \leftarrow \mathrm{NewtonSchulz5}(G_t)$
7: Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta(O_t + \lambda\theta_{t-1})$
8: **end for**
9: **return** θ_t

Newton-Schulz вместо SVD для вычисления $UV^\top$:

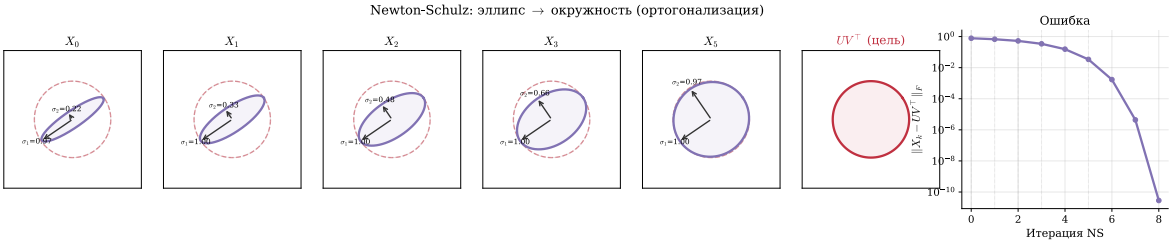
$$X_0 = G/\|G\|_F$$

$$X_{k+1} = aX_k + bX_kX_k^\top X_k + c(X_kX_k^\top)^2X_k$$

с коэффициентами a=3.4445, b=−4.7750, c=2.0315. ??? уточнить

- **5 итераций** достаточно на практике
- Только матричные умножения — эффективно на GPU
- Кубическая сходимость к полярному фактору

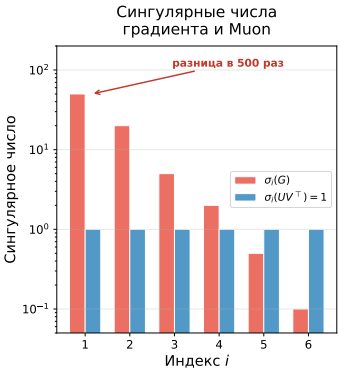
Newton-Schulz: эллипс → окружность

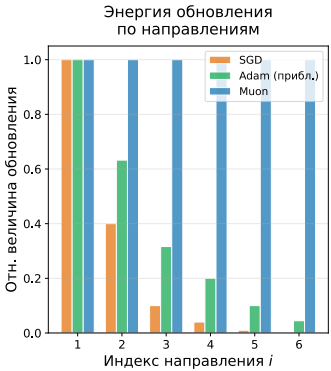


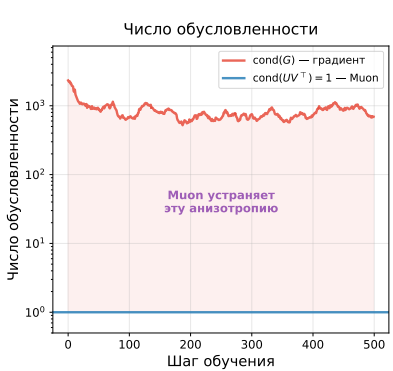
Классическая итерация $X_{k+1} = \frac{1}{2} X_k (3I - X_k^\top X_k)$: произвольная матрица G отображает окружность в эллипс. Newton-Schulz итеративно «округляет» эллипс, выравнивая сингулярные числа $\sigma_i(X_k) \to 1$. За 5–8 шагов ошибка $\|X_k - UV^\top\|_F$ падает до 10^{-10}.

Результаты и эксперименты

Muon: геометрия ортогонализации

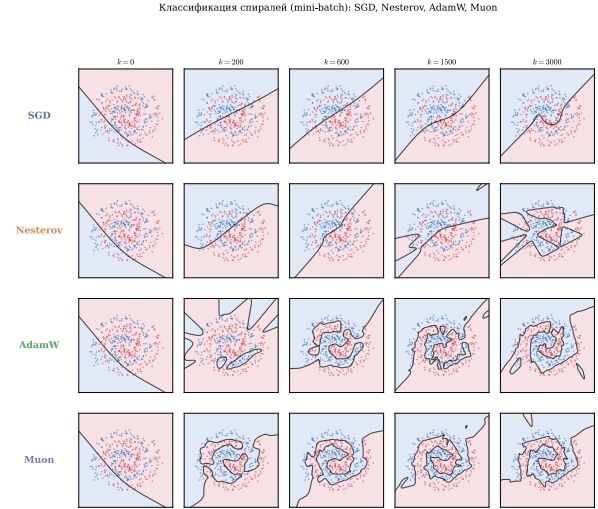


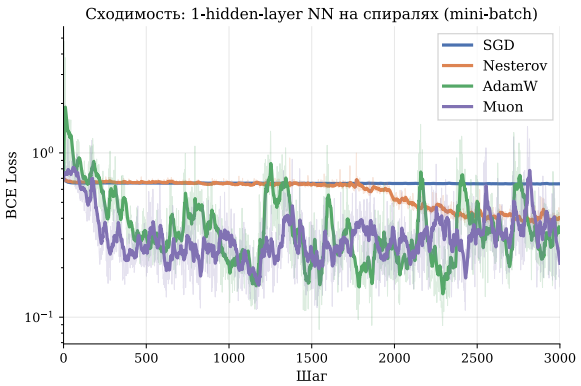




Вывод: SGD концентрирует обновление на доминирующих сингулярных направлениях (разница $500\times$). Muon нормализует все направления к единице — **равномерный прогресс** во всех направлениях пространства параметров.

SGD vs Nesterov vs AdamW vs Muon: классификация спиралей

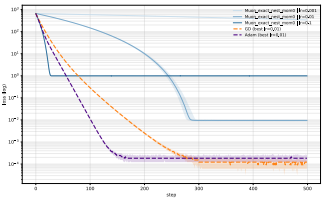


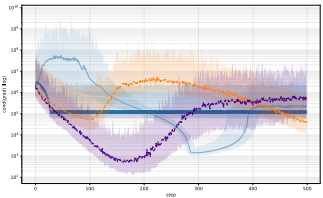


Обучение 1-hidden-layer NN (64 нейрона, mini-batch=40, 3000 шагов). Muon применяется к матричному параметру $W_1 \in \mathbb{R}^{64\times 3}$, AdamW — к выходному слою.

- Визуализация и Optuna HP tuning

Muon на квадратичных задачах [17]

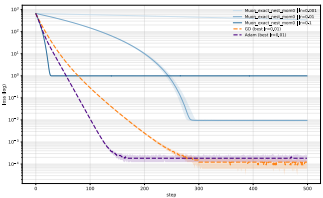


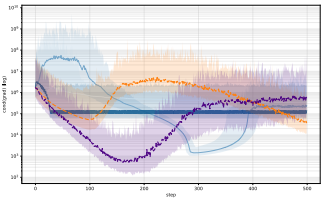


$L(W) = \frac{1}{2}\|W\|_F^2$, exact Stiefel/polar projection, без моментума.

- Динамика сводится к d скалярных рекурсий: $s_{t+1} = s_t - \alpha\,\mathrm{sign}(s_t)$

Muon на квадратичных задачах [17]

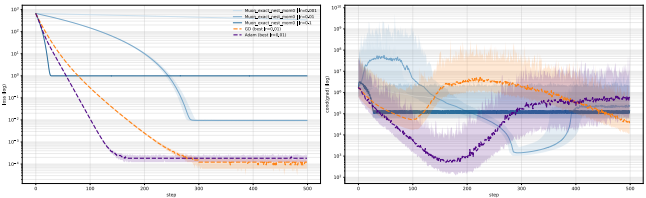




$L(W) = \frac{1}{2}\|W\|_F^2$, exact Stiefel/polar projection, без моментума.

- Динамика сводится к d скалярных рекурсий: $s_{t+1} = s_t - \alpha\,\mathrm{sign}(s_t)$
- Траектория остаётся на решётке $s_0 + \alpha\mathbb{Z} \to$ **2-цикл** около нуля (grid confinement)

Muon на квадратичных задачах [17]

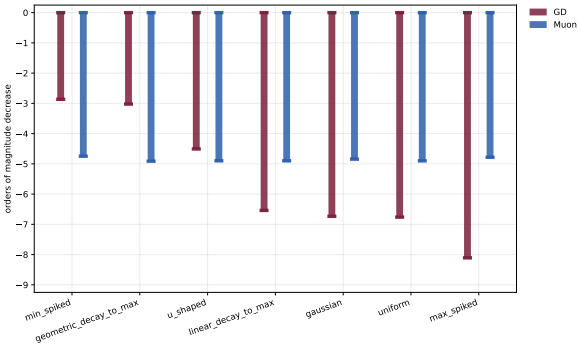


$L(W) = \frac{1}{2}\|W\|_F^2$, exact Stiefel/polar projection, без моментума.

- Динамика сводится к d скалярных рекурсий: $s_{t+1} = s_t - \alpha\,\mathrm{sign}(s_t)$
- Траектория остаётся на решётке $s_0 + \alpha\mathbb{Z} \to$ **2-цикл** около нуля (grid confinement)
- Loss floor: $\Theta(\alpha^2)$. Чтобы достичь ε: $O(1/\sqrt{\varepsilon})$ шагов vs $O(\log(1/\varepsilon))$ у GD

[17] Gonon, Muşat, Boumal (2026). Insights on Muon from Simple Quadratics.

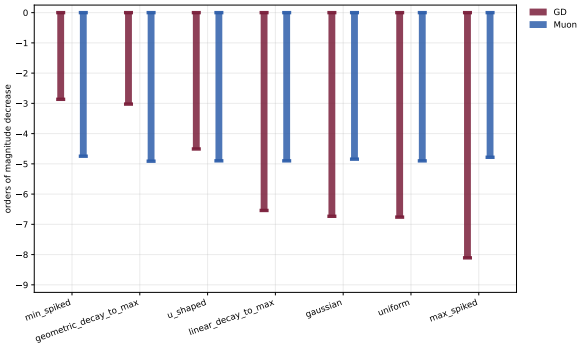
Muon: стабильность по спектру задачи [18]



7 типов распределения собственных значений, одинаковые $(s_{\min}, s_{\max}) = (10^{-3}, 10)$, $\kappa = 10^4$, $T = 500$.

- **Muon**: стабильно ≈ 5 порядков снижения на **всех** спектрах

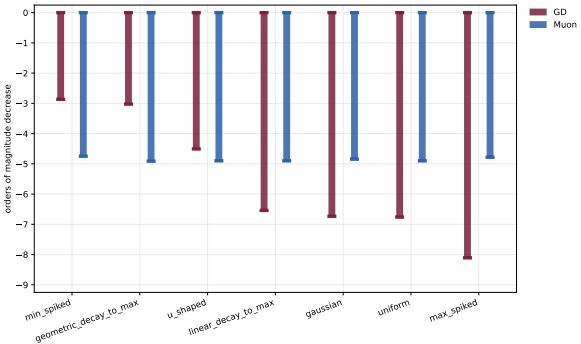
Muon: стабильность по спектру задачи [18]



7 типов распределения собственных значений, одинаковые $(s_{\min}, s_{\max}) = (10^{-3}, 10)$, $\kappa = 10^4$, $T = 500$.

- **Muon**: стабильно ≈ 5 порядков снижения на **всех** спектрах
- **GD**: от 3 до 8+ порядков — **сильно зависит** от формы спектра

Muon: стабильность по спектру задачи [18]

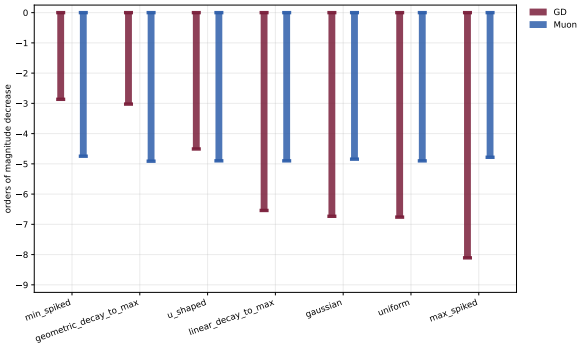


7 типов распределения собственных значений, одинаковые $(s_{\min}, s_{\max}) = (10^{-3}, 10)$, $\kappa = 10^4$, $T = 500$.

- **Muon**: стабильно ≈ 5 порядков снижения на **всех** спектрах
- **GD**: от 3 до 8+ порядков — **сильно зависит** от формы спектра

[18] Gonon, Muşat, Boumal (2026)

Muon: стабильность по спектру задачи [18]



7 типов распределения собственных значений, одинаковые $(s_{\min}, s_{\max}) = (10^{-3}, 10)$, $\kappa = 10^4$, $T = 500$.

- **Muon**: стабильно ≈ 5 порядков снижения на **всех** спектрах
- **GD**: от 3 до 8+ порядков — **сильно зависит** от формы спектра

Главный вывод (цитата): *«conditioning alone does not determine which method is faster»*. При одном κ смена формы спектра **переворачивает** ранжирование Muon vs GD. Спектры с массой мелких собственных значений и немногими крупными (`min_spiked`, `geometric_decay_to_max`) благоприятны для Muon — и похожи на спектры гессианов в нейросетях.

[18] Gonon, Muşat, Boumal (2026)

Magma: случайное маскирование обновлений [19]

Идея: случайное пропускание обновлений параметров во время обучения LLM **улучшает** результат!

SkipUpdate: на каждом шаге блокируем обновление блока параметров с вероятностью $1 - p$, масштабируя оставшиеся на $1/p$.

Magma: случайное маскирование обновлений [19]

Идея: случайное пропускание обновлений параметров во время обучения LLM **улучшает** результат!

SkipUpdate: на каждом шаге блокируем обновление блока параметров с вероятностью $1-p$, масштабируя оставшиеся на $1/p$.

Magma (Momentum-Aligned Gradient Masking): модулирует маскирование через выравнивание с моментумом:

$$\tilde{s}_t^{(b)} = \text{sigmoid}\left(\frac{\cos(\mu_t^{(b)}, g_t^{(b)})}{\tau}\right)$$

[19] Joo, T. et al. (2026). On Surprising Effectiveness of Masking Updates in Adaptive Optimizers.

Magma: случайное маскирование обновлений [19]

Идея: случайное пропускание обновлений параметров во время обучения LLM **улучшает** результат!

SkipUpdate: на каждом шаге блокируем обновление блока параметров с вероятностью $1-p$, масштабируя оставшиеся на $1/p$.

Magma (Momentum-Aligned Gradient Masking): модулирует маскирование через выравнивание с моментумом:

$$\tilde{s}_t^{(b)} = \text{sigmoid}\left(\frac{\cos(\mu_t^{(b)}, g_t^{(b)})}{\tau}\right)$$

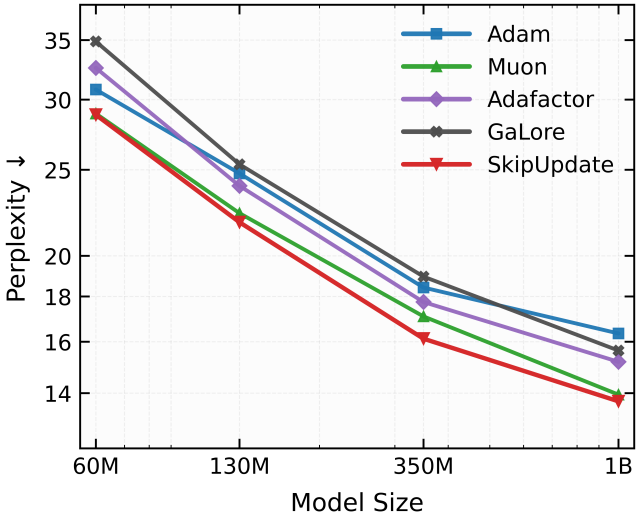
Теоретический инсайт: маскирование индуцирует геометрическую регуляризацию, зависящую от кривизны:

$$\mathscr{R}_t^{(b)} = \frac{1-p}{2p} \cdot (\Delta_t^{(b)})^\top H_{bb}(\theta_t) \Delta_t^{(b)}$$

Штрафует обновления вдоль направлений с большой кривизной — оптимизация смещается к более плоским минимумам.

[19] Joo, T. et al. (2026). On Surprising Effectiveness of Masking Updates in Adaptive Optimizers.

Magma: масштабирование до 1B параметров [20]



Вывод: SkipUpdate+Damping (Magma) стабильно превосходит Adam, Muon, Adafactor и GaLore на всех масштабах от 60M до 1B. На 1B параметрах: perplexity 13.8 vs 16.3 (Adam) — улучшение на **19%**.

[20] Joo et al. (2026)

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).
- Ключевой инсайт: ортогонализация ($UV^\top$) устраняет зависимость от числа обусловленности — **равномерный прогресс** по всем сингулярным направлениям.

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).
- Ключевой инсайт: ортогонализация ($UV^\top$) устраняет зависимость от числа обусловленности — **равномерный прогресс** по всем сингулярным направлениям.
- Newton-Schulz позволяет эффективно вычислять полярный фактор: 5 итераций матричных умножений вместо полного SVD.

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).
- Ключевой инсайт: ортогонализация ($UV^\top$) устраняет зависимость от числа обусловленности — **равномерный прогресс** по всем сингулярным направлениям.
- Newton-Schulz позволяет эффективно вычислять полярный фактор: 5 итераций матричных умножений вместо полного SVD.

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).
- Ключевой инсайт: ортогонализация ($UV^\top$) устраняет зависимость от числа обусловленности — **равномерный прогресс** по всем сингулярным направлениям.
- Newton-Schulz позволяет эффективно вычислять полярный фактор: 5 итераций матричных умножений вместо полного SVD.
- **На практике**: Muon — лучший по NanoGPT speedrun; используется в моделях масштаба 1T (Kimi K2).

Итоги лекции

- **Muon** — метод ортогонализации градиента, делающий **наискорейший спуск в спектральной норме** для матричных параметров.
- Shampoo приближает полноматричный предобусловитель AdaGrad через произведения Кронекера; Muon — его предельный случай (один градиент).
- Ключевой инсайт: ортогонализация ($UV^\top$) устраняет зависимость от числа обусловленности — **равномерный прогресс** по всем сингулярным направлениям.
- Newton-Schulz позволяет эффективно вычислять полярный фактор: 5 итераций матричных умножений вместо полного SVD.
- **На практике**: Muon — лучший по NanoGPT speedrun; используется в моделях масштаба 1T (Kimi K2).
- **Magma** — регуляризация через маскирование, смещающая обучение к плоским минимумам.

Дополнительные материалы

- J. Bernstein "Deriving Muon"

Дополнительные материалы

- J. Bernstein "Deriving Muon"
- K. Jordan "Muon: An optimizer for hidden layers"

Дополнительные материалы

- J. Bernstein "Deriving Muon"
- K. Jordan "Muon: An optimizer for hidden layers"
- Gonon et al. "Insights on Muon from Simple Quadratics" (2026)

Дополнительные материалы

- J. Bernstein "Deriving Muon"
- K. Jordan "Muon: An optimizer for hidden layers"
- Gonon et al. "Insights on Muon from Simple Quadratics" (2026)
- Joo et al. "On Surprising Effectiveness of Masking Updates" (2026)

Дополнительные материалы

- J. Bernstein "Deriving Muon"
- K. Jordan "Muon: An optimizer for hidden layers"
- Gonon et al. "Insights on Muon from Simple Quadratics" (2026)
- Joo et al. "On Surprising Effectiveness of Masking Updates" (2026)
- Д. Ковалёв "Understanding Gradient Orthogonalization" (теория Muon)