Стохастический градиентный спуск. Адаптивные методы. Оптимизация нейронных сетей

Семинар

Оптимизация для всех! ЦУ

Задача конечной суммы

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

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

Градиентный спуск действует следующим образом:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}$$

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

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

Градиентный спуск действует следующим образом:

$$
x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}
$$

- Сходимость с постоянным α или линейным поиском.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

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

Градиентный спуск действует следующим образом:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

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

Градиентный спуск действует следующим образом:

$$
x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_i(x) \tag{GD}
$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Задача конечной суммы

Рассмотрим классическую задачу минимизации среднего на конечной выборке:

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

Градиентный спуск действует следующим образом:

$$x_{k+1} = x_k - \frac{\alpha_k}{n} \sum_{i=1}^n \nabla f_i(x) \tag{GD}$$

- Сходимость с постоянным α или линейным поиском.
- Стоимость итерации линейна по n. Для ImageNet $n \approx 1.4 \cdot 10^7$, для WikiText $n \approx 10^8$.

Перейдем от вычисления полного градиента к его несмещенной оценке, когда мы случайно выбираем индекс i_k точки на каждой итерации равномерно:

$$x_{k+1} = x_k - \alpha_k \nabla f_{i_k}(x_k) \tag{SGD}$$

При $p(i_k = i) = \frac{1}{n}$ стохастический градиент является несмещенной оценкой градиента, определяемой как:

$$\mathbb{E}[\nabla f_{i_k}(x)] = \sum_{i=1}^n p(i_k = i) \nabla f_i(x) = \sum_{i=1}^n \frac{1}{n} \nabla f_i(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x) = \nabla f(x)$$

Это указывает на то, что математическое ожидание стохастического градиента равно фактическому градиенту

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.
    - Оценки неулучшаемы при стандартных предположениях.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.
    - Оценки неулучшаемы при стандартных предположениях.
    - Оракул возвращает несмещенную аппроксимацию градиента с ограниченной дисперсией.

Результаты для градиентного спуска

Стохастические итерации в n раз быстрее, но сколько итераций требуется?

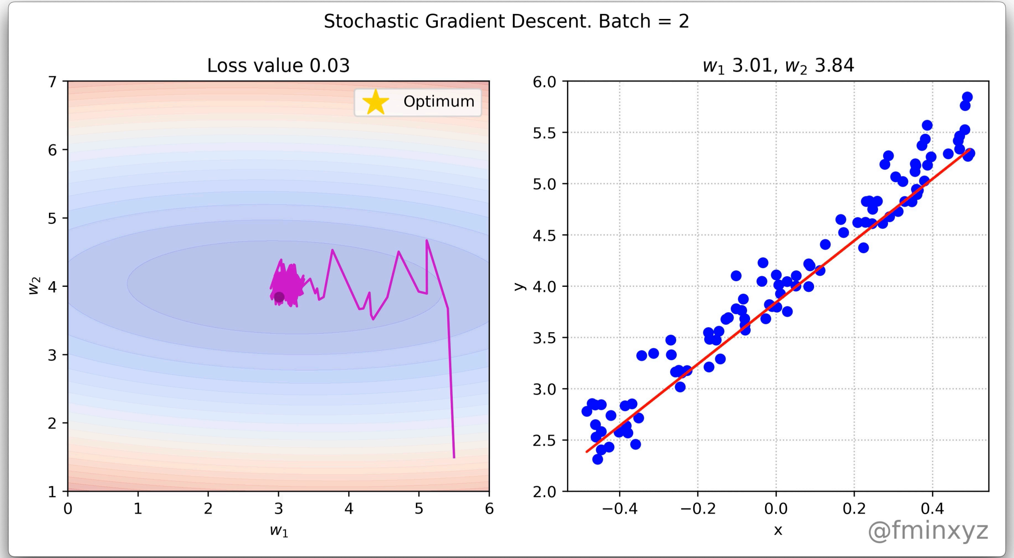
Если ∇f является Липшицевым, то мы имеем:

Предположение	Детерминированный градиентный спуск	Стохастический градиентный спуск
PL	$O(\log(1/\varepsilon))$	$O(1/\varepsilon)$
Выпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$
Невыпуклая	$O(1/\varepsilon)$	$O(1/\varepsilon^2)$

- Стохастический метод имеет низкую стоимость итерации, но медленную скорость сходимости.
    - Сублинейная скорость даже в сильно выпуклом случае.
    - Оценки неулучшаемы при стандартных предположениях.
    - Оракул возвращает несмещенную аппроксимацию градиента с ограниченной дисперсией.
- Momentum и квазиньютоновские методы не улучшают скорость сходимости в стохастическом случае. Могут улучшить только константы (узким местом является дисперсия, а не число обусловленности).

Стохастический градиентный спуск (SGD)

Типичное поведение



Вычислительные эксперименты

Вычислительные эксперименты

Визуализация SGD.

Посмотрим на вычислительные эксперименты для SGD.

Адаптивность или масштабирование

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

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

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

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \ldots, p$:

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

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

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \ldots, p$:

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

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

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

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

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

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \ldots, p$:

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

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

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.
- Основным недостатком является монотонное накопление градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны при обучении глубоких нейронных сетей.

Adagrad (Duchi, Hazan, and Singer 2010/Streeter and MacMahan 2010)

Очень популярный адаптивный метод. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$, и обновление для $j = 1, \dots, p$:

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

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

Заметки:

- AdaGrad не требует настройки скорости обучения: $\alpha > 0$ — фиксированная константа, и скорость обучения естественным образом уменьшается с итерациями.
- Скорость обучения редких информативных признаков уменьшается медленно.
- Может значительно превосходить SGD в разреженных задачах.
- Основным недостатком является монотонное накопление градиентов в знаменателе. AdaDelta, Adam, AMSGrad и др. улучшают это, популярны при обучении глубоких нейронных сетей.
- Константа ϵ обычно устанавливается равной 10^{-6}, чтобы избежать деления на ноль или слишком больших шагов.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

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

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

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

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

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

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \ldots, p$:

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

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

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.
- Позволяет более тонко настраивать скорость обучения, чем AdaGrad, что делает его подходящим для неизотропных задач.

RMSProp (Tieleman and Hinton, 2012)

Улучшение AdaGrad, решающее проблему агрессивного, монотонно убывающего темпа обучения. Использует скользящее среднее квадратов градиентов для настройки скорости обучения для каждого веса. Пусть $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$ и правило обновления для $j = 1, \dots, p$:

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

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

Заметки:

- RMSProp делит скорость обучения для веса на скользящее среднее величин последних градиентов для этого веса.
- Позволяет более тонко настраивать скорость обучения, чем AdaGrad, что делает его подходящим для неизотропных задач.
- Часто используется при обучении нейронных сетей, особенно рекуррентных.

Adam (Kingma and Ba, 2014) [1 2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

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

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

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

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

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

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

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

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

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

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

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

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

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

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

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

Исправление смещения:

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

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

Обновление:

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

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

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

Исправление смещения:

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

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

Обновление:

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)
- Каким-то образом работает исключительно хорошо для некоторых сложных задач

Adam (Kingma and Ba, 2014) [1] [2]

Сочетает в себе элементы как AdaGrad, так и RMSProp. Рассматривает экспоненциально затухающее среднее прошлых градиентов и квадратов градиентов.

EMA:

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

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

Исправление смещения:

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

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

Обновление:

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

Заметки:

- Он исправляет смещение к нулю в начальные моменты, наблюдаемое в других методах, таких как RMSProp, делая оценки более точными.
- Одна из самых цитируемых научных статей в мире
- В 2018-2019 годах были опубликованы статьи, указывающие на ошибки в оригинальной статье
- Не сходится на некоторых простых задачах (даже выпуклых)
- Каким-то образом работает исключительно хорошо для некоторых сложных задач
- Работает намного лучше для языковых моделей, чем для задач компьютерного зрения — почему?

[1] Adam: A Method for Stochastic Optimization

[2] On the Convergence of Adam and Beyond

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda \|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$
\begin{aligned}
m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\
v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2 \\
\hat{m}_j &= \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k} \\
x_j^{(k)} &= x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)
\end{aligned}
$$

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$
\begin{aligned}
m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\
v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2)(g_j^{(k)})^2 \\
\hat{m}_j &= \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k} \\
x_j^{(k)} &= x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)
\end{aligned}
$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется *после* шага адаптивного градиента.

AdamW (Loshchilov & Hutter, 2017)

Решает распространенную проблему с ℓ_2 регуляризацией в адаптивных оптимизаторах, таких как Adam. Стандартная ℓ_2 регуляризация добавляет $\lambda\|x\|^2$ к функции потерь, что приводит к слагаемому градиента λx. В Adam это слагаемое масштабируется адаптивной скоростью обучения $\left(\sqrt{\hat{v}_j} + \epsilon\right)$, связывая затухание весов с величинами градиента.

AdamW отделяет затухание весов от шага адаптации градиента.

Правило обновления:

$$
\begin{aligned}
m_j^{(k)} &= \beta_1 m_j^{(k-1)} + (1 - \beta_1) g_j^{(k)} \\
v_j^{(k)} &= \beta_2 v_j^{(k-1)} + (1 - \beta_2) (g_j^{(k)})^2 \\
\hat{m}_j &= \frac{m_j^{(k)}}{1 - \beta_1^k}, \quad \hat{v}_j = \frac{v_j^{(k)}}{1 - \beta_2^k} \\
x_j^{(k)} &= x_j^{(k-1)} - \alpha \left(\frac{\hat{m}_j}{\sqrt{\hat{v}_j} + \epsilon} + \lambda x_j^{(k-1)} \right)
\end{aligned}
$$

Заметки:

- Слагаемое затухания весов $\lambda x_j^{(k-1)}$ добавляется *после* шага адаптивного градиента.
- Широко применяется при обучении трансформеров и других больших моделей. Выбор по умолчанию для huggingface trainer.

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

1. Вычислить градиент G_k.

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Заметки:

- Нацелен на более эффективный учет информации о кривизне, чем методы первого порядка.

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Заметки:

- Нацелен на более эффективный учет информации о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или к лучшим решениям с точки зрения количества шагов.

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

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

Заметки:

- Нацелен на более эффективный учет информации о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или к лучшим решениям с точки зрения количества шагов.
- Требует тщательной реализации для эффективности (например, эффективное вычисление обратных матричных корней, работа с большими матрицами).

Shampoo [3]

Расшифровывается как **S**tochastic **H**essian-**A**pproximation **M**atrix **P**reconditioning for **O**ptimization **O**f deep networks. Это метод, вдохновленный оптимизацией второго порядка, разработанный для глубокого обучения больших моделей.

Основная идея: Аппроксимирует полноматричный предобуславливатель AdaGrad, используя эффективные матричные структуры, в частности произведения Кронекера.

Для матрицы весов $W \in \mathbb{R}^{m \times n}$ обновление включает предобуславливание с использованием аппроксимаций матриц статистики $L \approx \sum_k G_k G_k^T$ и $R \approx \sum_k G_k^T G_k$, где G_k — градиенты.

Упрощенная концепция:

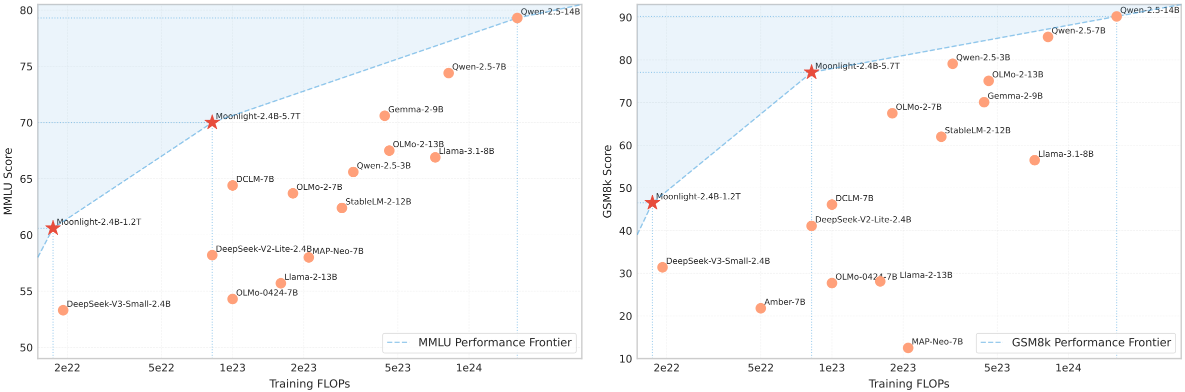
1. Вычислить градиент G_k.
2. Обновить статистики $L_k = \beta L_{k-1} + (1-\beta) G_k G_k^T$ и $R_k = \beta R_{k-1} + (1-\beta) G_k^T G_k$.
3. Вычислить предобуславливатели $P_L = L_k^{-1/4}$ и $P_R = R_k^{-1/4}$. (Обратный матричный корень)
4. Обновить: $W_{k+1} = W_k - \alpha P_L G_k P_R$.

Заметки:

- Нацелен на более эффективный учет информации о кривизне, чем методы первого порядка.
- Вычислительно дороже, чем Adam, но может сходиться быстрее или к лучшим решениям с точки зрения количества шагов.
- Требует тщательной реализации для эффективности (например, эффективное вычисление обратных матричных корней, работа с большими матрицами).

Muon

Новый подход к оптимизации [4]



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

[4] KIMI K2: OPEN AGENTIC INTELLIGENCE

Интуиция за методом Muon [5]

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

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

[5] Презентация R. Gower

Интуиция за методом Muon [5]

Функция потерь

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

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

[5] Презентация R. Gower

Интуиция за методом Muon [5]

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 [5]

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

[5] Презентация R. Gower

Интуиция за методом Muon. Градиентный спуск

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

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

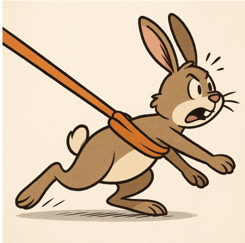
$$
\begin{aligned}
x_{k+1} &= \operatorname*{argmin}_{x \in \mathbb{R}^p} \left(f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{2\alpha} \|x - x_k\|_2^2 \right) \\
&= x_k - \alpha \nabla f(x_k)
\end{aligned}
$$

Интуиция за методом Muon. Градиентный спуск

Штраф за дальность от x_k

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

Шаг обучения / коэффициент регуляризации



Интуиция за методом Muon. Нормированный градиентный спуск

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

Интуиция за методом Muon. Нормированный градиентный спуск

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

Интуиция за методом Muon. Нормированный градиентный спуск

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

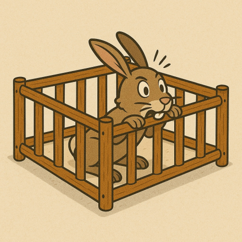
Ограничение на длину шага

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?

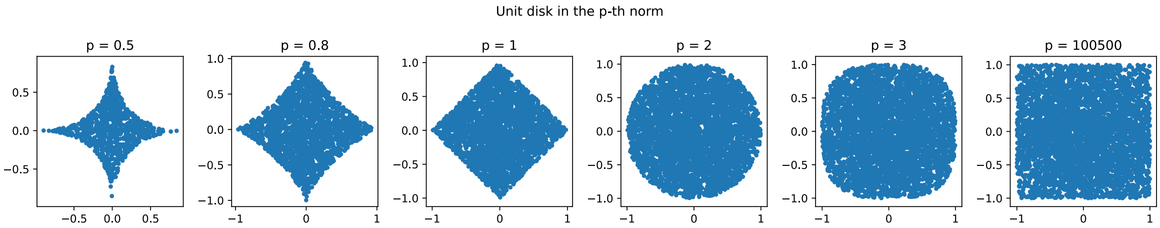


Рис. 2: Примеры шаров в разных нормах

Неевклидовы записи методов [6]

Linear Minimization Oracle:

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

Неевклидов градиентный спуск

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

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

[6] Old Optimizer, New Norm: An Anthology

Неевклидовы записи методов [6]

Linear Minimization Oracle:

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

Неевклидов градиентный спуск

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

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

Неевклидов нормированный градиентный спуск

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

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

[6] 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 длины.

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 $\|\cdot\|_{\mathrm{op}}$, ядерную $\|\cdot\|_{\mathrm{nuc}}$, Фробениуса $\|\cdot\|_F$ и т.п.
- Вся логика переносится: вместо вектора ищем «лучшее направление спуска» среди матриц заданной длины.
- Скалярное произведение:

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

Неевклидов нормированный спуск для матриц

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

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

где

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

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

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

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

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

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

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

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

  то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k.

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

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

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

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

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

  то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k.
- Проблема: полное SVD на каждом шаге дорого. Хорошая новость: нам нужен только $(UV^\top)$, его можно считать гораздо быстрее:

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

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

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

— редуцированное SVD градиента. Тогда

- LMO (с «max»-формулировкой) по операторной норме:

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

  то есть оптимальное направление — **polar factor** (matrix sign) матрицы G_k.
- Проблема: полное SVD на каждом шаге дорого. Хорошая новость: нам нужен только $(UV^\top)$, его можно считать гораздо быстрее:
- итерациями **Newton–Schulz**/ **Polar Express**, которые используют только матричные умножения, дают приближение $UV^\top$ за несколько шагов и снимают узкое место полного SVD внутри Muon.

Обучение GPT-2 (124M) на FineWeb

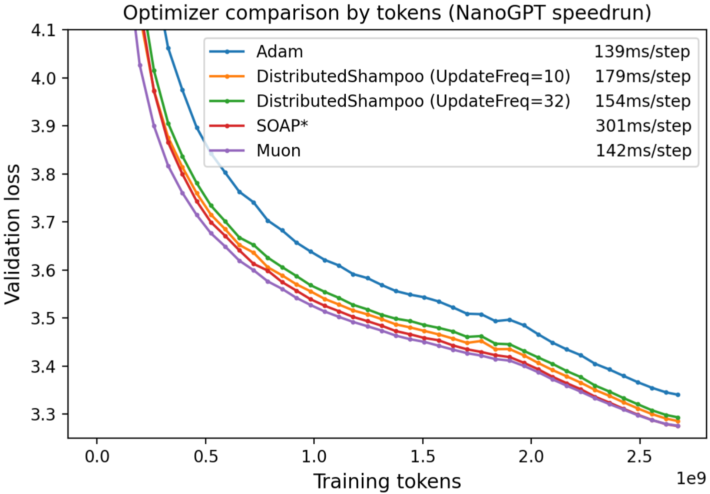


Рис. 3: NanoGPT speedrun

Обучение GPT-2 (124M) на FineWeb

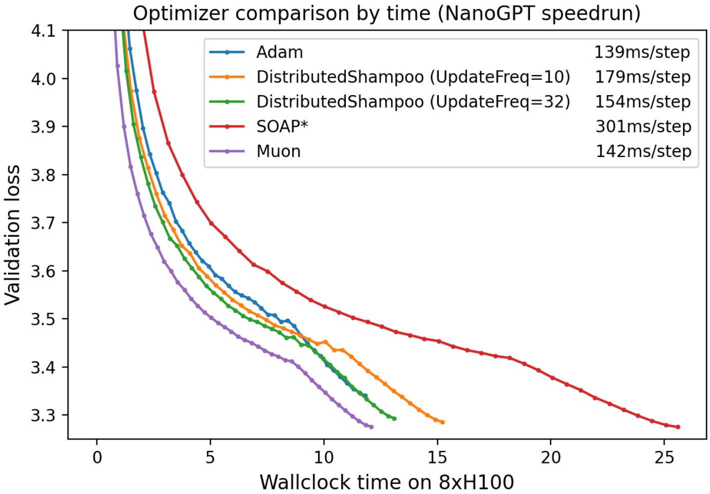


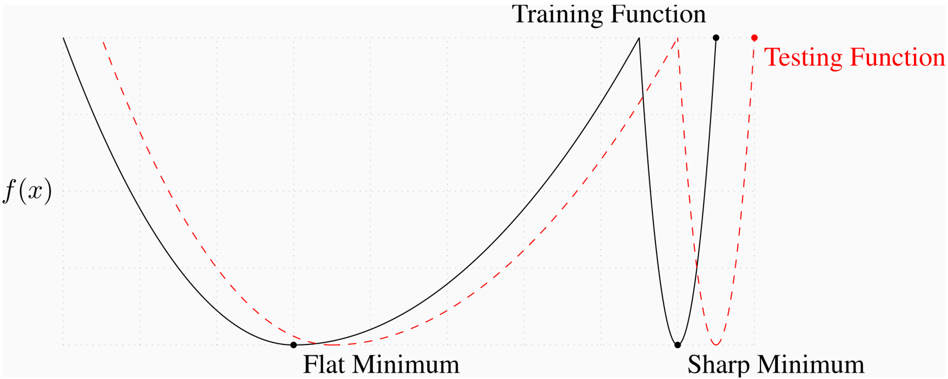
Рис. 4: NanoGPT speedrun

Сравнение Muon с AdamW на LogReg

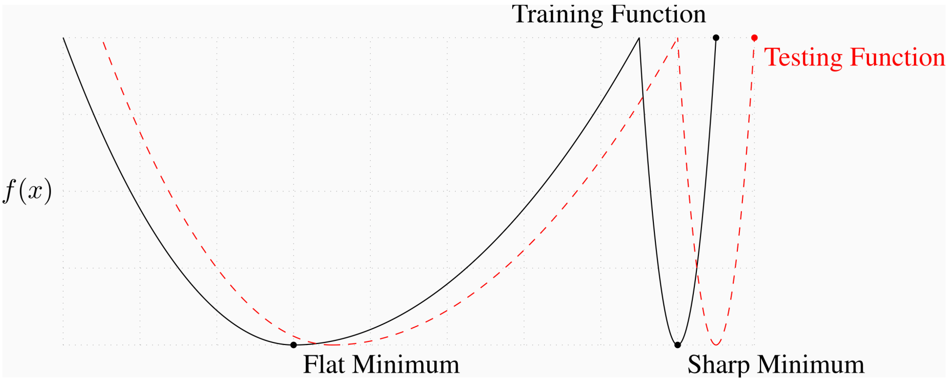
Простое сравнение Muon и AdamW на небольшой задаче LogReg

SAM

Плоский минимум vs Острый минимум



Плоский минимум vs Острый минимум



Question

Что не так с острым минимумом?

Sharpness-Aware Minimization [7]

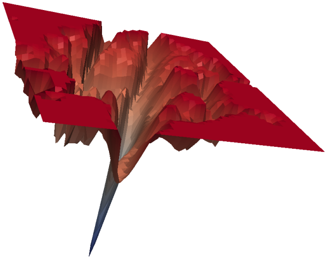


Рис. 5: Острый минимум, к которому сошлась ResNet, обученная с помощью SGD.

[7] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Sharpness-Aware Minimization [7]

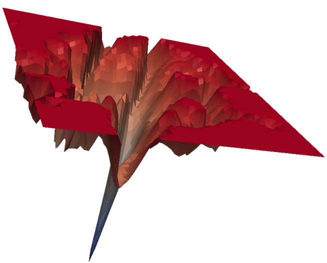


Рис. 5: Острый минимум, к которому сошлась ResNet, обученная с помощью SGD.

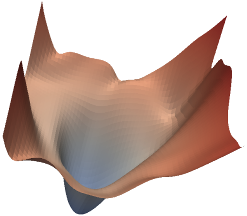


Рис. 6: Широкий минимум, к которому сошлась та же ResNet, обученная с помощью SAM.

[7] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Sharpness-Aware Minimization [7]

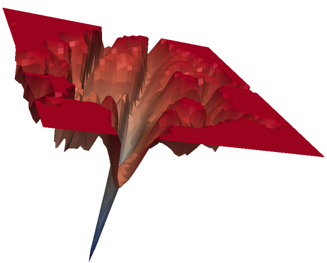


Рис. 5: Острый минимум, к которому сошлась ResNet, обученная с помощью SGD.

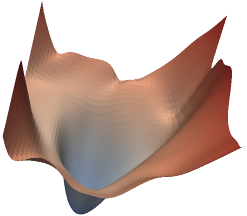


Рис. 6: Широкий минимум, к которому сошлась та же ResNet, обученная с помощью SAM.

Sharpness-Aware Minimization (SAM) — это процедура, целью которой является улучшение обобщающей способности модели путем одновременной минимизации значения функции потерь и **остроты (sharpness) функции потерь**.

[7] Foret, Pierre, et al. "Sharpness-aware minimization for efficiently improving generalization." (2020)

Постановка задачи обучения

Обучающая выборка, полученная $i.i.d.$ из распределения D:

$$S = \{(x_i, y_i)\}_{i=1}^n,$$

где x_i — вектор признаков, а y_i — метка.

Постановка задачи обучения

Обучающая выборка, полученная $i.i.d.$ из распределения D:

$$S = \{(x_i, y_i)\}_{i=1}^n,$$

где x_i — вектор признаков, а y_i — метка.

Функция потерь на обучающей выборке:

$$L_S = \frac{1}{n}\sum_{i=1}^n l(w, x_i, y_i),$$

где l — функция потерь для одного объекта, w — параметры.

Постановка задачи обучения

Обучающая выборка, полученная $i.i.d.$ из распределения D:

$$S = \{(x_i, y_i)\}_{i=1}^n,$$

где x_i — вектор признаков, а y_i — метка.

Функция потерь на обучающей выборке:

$$L_S = \frac{1}{n}\sum_{i=1}^n l(w, x_i, y_i),$$

где l — функция потерь для одного объекта, w — параметры.

Функция потерь на генеральной совокупности (population loss):

$$L_D = \mathbb{E}_{(x,y)}[l(w, x, y)]$$

Что такое sharpness (острота)?

Theorem

Для любого $\rho > 0$, с высокой вероятностью по обучающей выборке S, сгенерированной из распределения D,

$$L_D(w) \leq \max_{\|\epsilon\|_2 \leq \rho} L_S(w + \epsilon) + h\left(\|w\|_2^2/\rho^2\right),$$

где $h : \mathbb{R}_+ \to \mathbb{R}_+$ — строго возрастающая функция (при некоторых технических условиях на $L_D(w)$).

Что такое sharpness (острота)?

Theorem

Для любого $\rho > 0$, с высокой вероятностью по обучающей выборке S, сгенерированной из распределения D,

$$L_D(w) \leq \max_{\|\epsilon\|_2 \leq \rho} L_S(w+\epsilon) + h\left(\|w\|_2^2/\rho^2\right),$$

где $h : \mathbb{R}_+ \to \mathbb{R}_+$ — строго возрастающая функция (при некоторых технических условиях на $L_D(w)$).

Добавляя и вычитая $L_S(w)$:

$$\left[\max_{\|\epsilon\|_2 \leq \rho} L_S(w+\epsilon) - L_S(w)\right] + L_S(w) + h\left(\|w\|_2^2/\rho^2\right)$$

Слагаемое в квадратных скобках отражает **остроту (sharpness)** L_S в точке w, измеряя, как быстро может возрасти ошибка обучения при переходе от w к близкому значению параметров.

Sharpness-Aware Minimization

Функция h заменяется на более простую константу λ. Авторы предлагают выбирать значения параметров, решая следующую задачу Sharpness-Aware Minimization (SAM):

$$\min_{w} L_S^{SAM}(w) + \lambda\|w\|_2^2 \quad \text{где} \quad L_S^{SAM}(w) \triangleq \max_{\|\epsilon\|_p \leq \rho} L_S(w+\epsilon),$$

с гиперпараметром $\rho \geq 0$ и p из $[1, \infty]$ (небольшое обобщение, хотя $p = 2$ эмпирически является лучшим выбором).

Как минимизировать L_S^{SAM}?

Для минимизации L_S^{SAM} используется эффективная аппроксимация его градиента. Первым шагом рассматривается разложение Тейлора первого порядка для $L_S(w+\epsilon)$:

$$\epsilon^*(w) \triangleq \underset{\|\epsilon\|_p \le \rho}{\arg\max} \left\{L_S(w+\epsilon)\right\} \approx \underset{\|\epsilon\|_p \le \rho}{\arg\max} \left\{L_S(w) + \epsilon^T \nabla_w L_S(w)\right\} = \underset{\|\epsilon\|_p \le \rho}{\arg\max} \left\{\epsilon^T \nabla_w L_S(w)\right\}.$$

Как минимизировать L_S^{SAM}?

Для минимизации L_S^{SAM} используется эффективная аппроксимация его градиента. Первым шагом рассматривается разложение Тейлора первого порядка для $L_S(w+\epsilon)$:

$$\epsilon^*(w) \triangleq \arg\max_{\|\epsilon\|_p \le \rho} \left\{ L_S(w+\epsilon) \right\} \approx \arg\max_{\|\epsilon\|_p \le \rho} \left\{ L_S(w) + \epsilon^T \nabla_w L_S(w) \right\} = \arg\max_{\|\epsilon\|_p \le \rho} \left\{ \epsilon^T \nabla_w L_S(w) \right\}.$$

Последнее выражение — это просто argmax скалярного произведения векторов ϵ и $\nabla_w L_S(w)$, и хорошо известно, какой аргумент его максимизирует:

$$\hat{\epsilon}(w) = \rho \operatorname{sign}\left(\nabla_w L_S(w)\right) \left|\nabla_w L_S(w)\right|^{q-1} / \left(\left\|\nabla_w L_S(w)\right\|_q^q \right)^{1/p},$$

где $1/p + 1/q = 1$.

Как минимизировать L_S^{SAM}?

Для минимизации L_S^{SAM} используется эффективная аппроксимация его градиента. Первым шагом рассматривается разложение Тейлора первого порядка для $L_S(w+\epsilon)$:

$$\epsilon^*(w) \triangleq \arg\max_{\|\epsilon\|_p \le \rho} \{L_S(w+\epsilon)\} \approx \arg\max_{\|\epsilon\|_p \le \rho} \{L_S(w) + \epsilon^T \nabla_w L_S(w)\} = \arg\max_{\|\epsilon\|_p \le \rho} \{\epsilon^T \nabla_w L_S(w)\}.$$

Последнее выражение — это просто argmax скалярного произведения векторов ϵ и $\nabla_w L_S(w)$, и хорошо известно, какой аргумент его максимизирует:

$$\hat{\epsilon}(w) = \rho \operatorname{sign}(\nabla_w L_S(w)) |\nabla_w L_S(w)|^{q-1} / \left(\|\nabla_w L_S(w)\|_q^q\right)^{1/p},$$

где $1/p + 1/q = 1$.

Таким образом

$$\nabla_w L_S^{SAM}(w) \approx \nabla_w L_S(w+\hat{\epsilon}(w)) = \frac{d(w+\hat{\epsilon}(w))}{dw} \nabla_w L_S(w)\Big|_{w+\hat{\epsilon}(w)}$$

$$= \nabla_w L_S(w)|_{w+\hat{\epsilon}(w)} + \frac{d\hat{\epsilon}(w)}{dw} \nabla_w L_S(w)\Big|_{w+\hat{\epsilon}(w)}$$

Sharpness-Aware Minimization

Современные фреймворки могут легко вычислить предыдущее приближение. Однако для ускорения вычислений члены второго порядка можно отбросить, получая:

$$\nabla_w L_S^{SAM}(w) \approx \nabla_w L_S(w)\Big|_{w+\hat{\epsilon}(w)}$$

Sharpness-Aware Minimization

Современные фреймворки могут легко вычислить предыдущее приближение. Однако для ускорения вычислений члены второго порядка можно отбросить, получая:

$$\nabla_w L_S^{SAM}(w) \approx \nabla_w L_S(w)\big|_{w+\hat{\epsilon}(w)}$$

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^n \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots (\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
  Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
  Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
  Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
  Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
  $t = t + 1$;
end
return $\boldsymbol{w}_t$

Algorithm 1: SAM algorithm

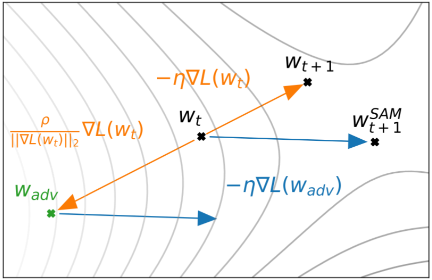


Figure 2: Schematic of the SAM parameter update.

Результаты SAM

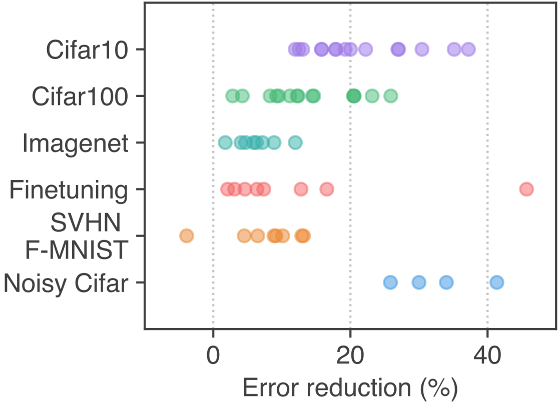


Рис. 8: Снижение частоты ошибок, полученное при переходе на SAM. Каждая точка соответствует отдельному набору данных / модели / аугментации данных.

Связность мод (Mode Connectivity)

Связность мод (Mode Connectivity) [8]

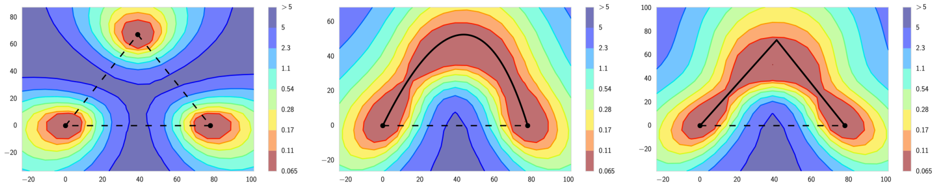


Рис. 9: Поверхность функции потерь (кросс-энтропия с l_2-регуляризацией) для ResNet-164 на CIFAR-100 как функция весов сети в двумерном подпространстве. На каждом графике горизонтальная ось фиксирована и проходит через оптимумы двух независимо обученных сетей. Вертикальная ось меняется между графиками при смене плоскостей (определенных в основном тексте). Слева: Три оптимума для независимо обученных сетей. В центре и справа: Квадратичная кривая Безье и ломаная с одним изгибом, соединяющие два нижних оптимума с левого графика вдоль пути с почти постоянной функцией потерь. Заметьте, что на каждом графике прямой линейный путь между модами привел бы к высоким потерям.

[8] Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D. P., Wilson, A. G. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. Advances in neural information processing systems, 31.

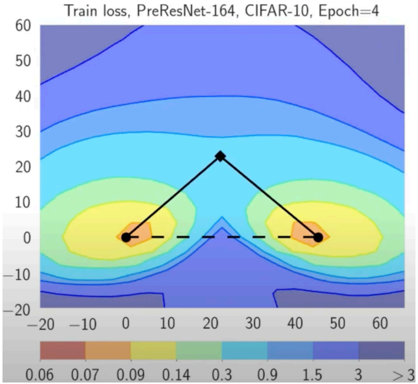
Процедура поиска кривой

- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t\sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

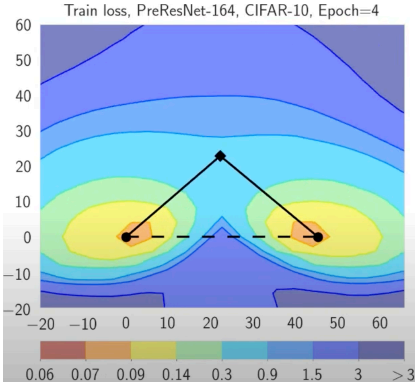
- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0, 1] \to \mathbb{R}^{|\text{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

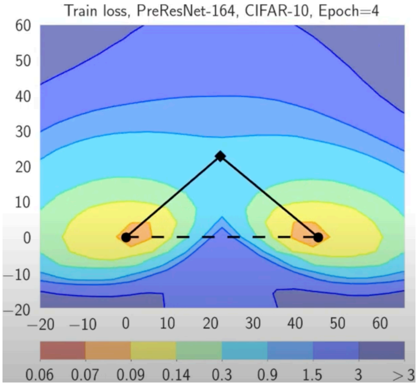
$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую:
  $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

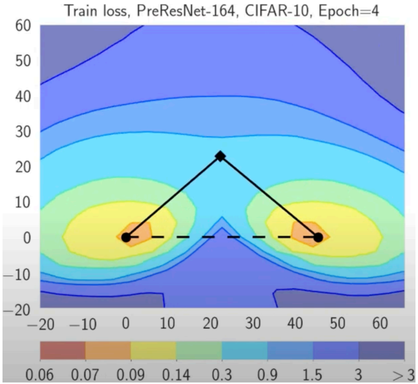
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$

- Минимизируем усредненную функцию потерь по θ:

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t\sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Процедура поиска кривой

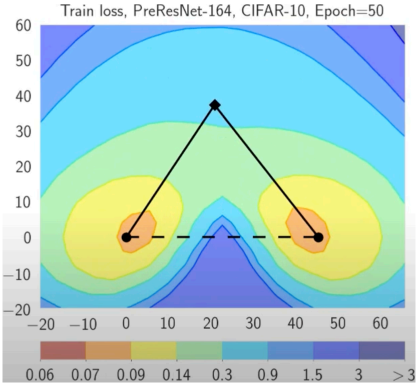
- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Процедура поиска кривой

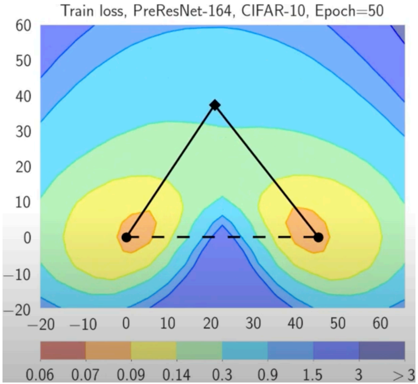
- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0, 1] \to \mathbb{R}^{|\text{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

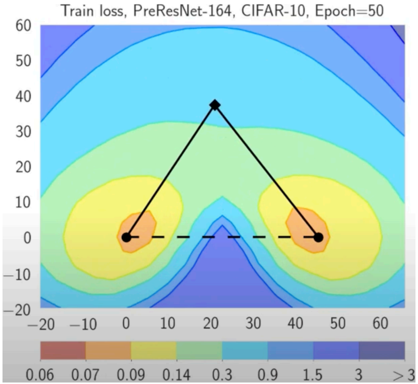
$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\mathsf{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\mathsf{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

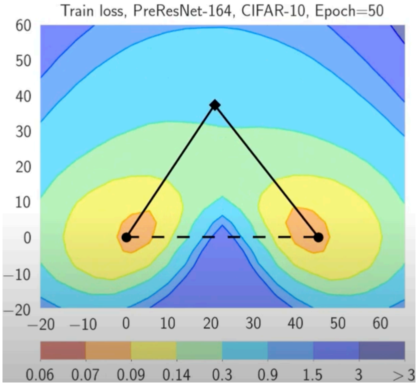
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$

- Минимизируем усредненную функцию потерь по θ:

$$\underset{\theta}{\text{minimize}} \ \ \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Процедура поиска кривой

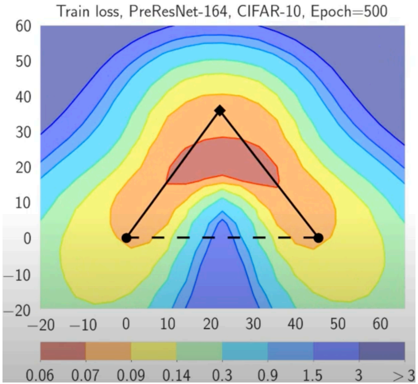
- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Процедура поиска кривой

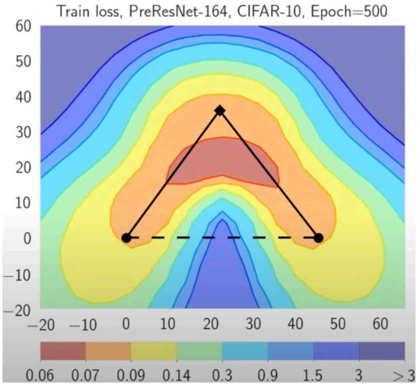
- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0, 1] \to \mathbb{R}^{|\text{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \ \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

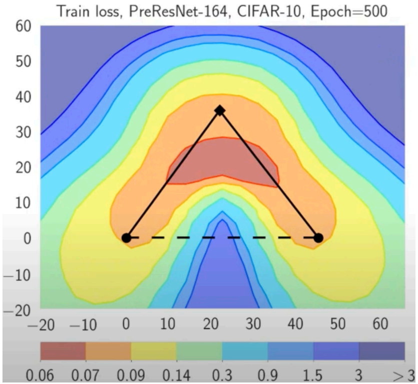
$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0, 1] \to \mathbb{R}^{|\text{net}|}$

$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$



$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$

Процедура поиска кривой

- Веса предобученных сетей:

$$\widehat{w}_1, \widehat{w}_2 \in \mathbb{R}^{|\text{net}|}$$

- Определим параметрическую кривую: $\phi_\theta(\cdot) : [0,1] \to \mathbb{R}^{|\text{net}|}$

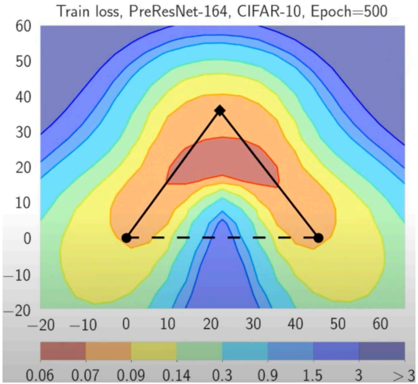
$$\phi_\theta(0) = \widehat{w}_1, \quad \phi_\theta(1) = \widehat{w}_2$$

- Функция потерь DNN:

$$\mathcal{L}(w)$$

- Минимизируем усредненную функцию потерь по θ:

$$\underset{\theta}{\text{minimize}} \;\; \ell(\theta) = \int_0^1 \mathcal{L}\left(\phi_\theta(t)\right) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}\left(\phi_\theta(t)\right)$$



Гроккинг (Grokking)

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание

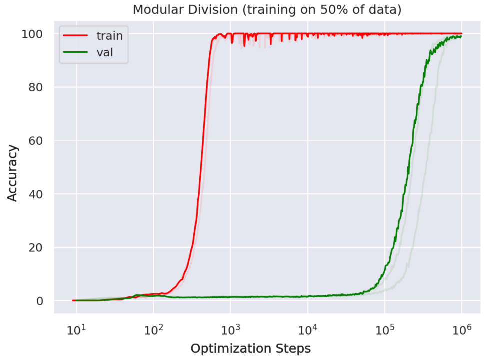


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание
- Возможно, они медленно дрейфуют к более широкому минимуму

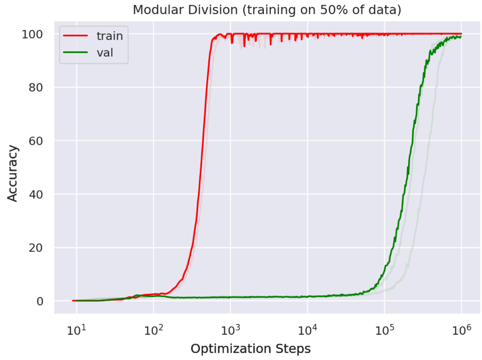


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание
- Возможно, они медленно дрейфуют к более широкому минимуму
- Недавно открытый эффект гроккинга подтверждает эту гипотезу

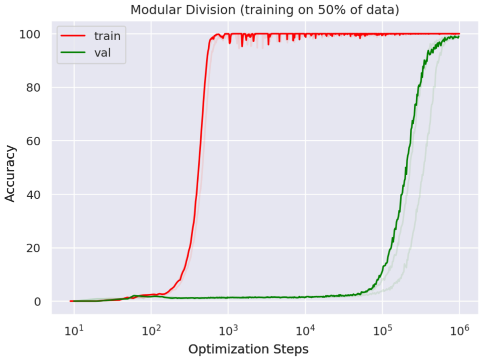


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание
- Возможно, они медленно дрейфуют к более широкому минимуму
- Недавно открытый эффект гроккинга подтверждает эту гипотезу
- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (*Surprising properties of loss landscape in overparameterized models*). видео, Презентация

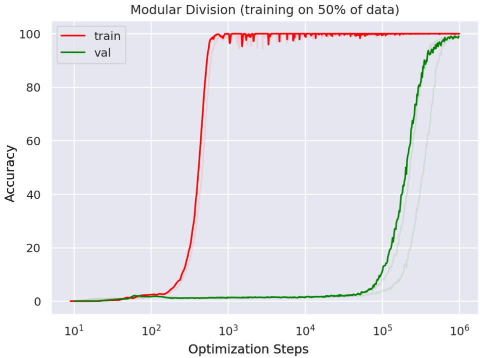


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание
- Возможно, они медленно дрейфуют к более широкому минимуму
- Недавно открытый эффект гроккинга подтверждает эту гипотезу
- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (*Surprising properties of loss landscape in overparameterized models*). видео, Презентация
- Автор канала Свидетели Градиента собирает интересные наблюдения и эксперименты про гроккинг.

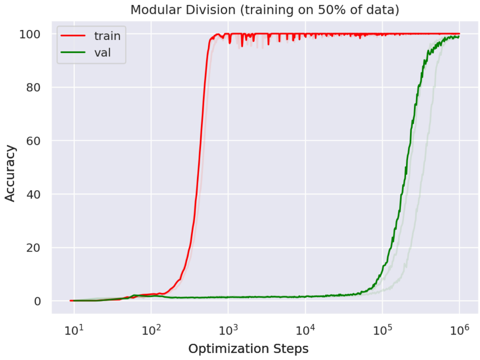


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

Гроккинг (Grokking)[9]

- После достижения нулевой ошибки на обучении веса продолжают изменяться в манере, напоминающей случайное блуждание
- Возможно, они медленно дрейфуют к более широкому минимуму
- Недавно открытый эффект гроккинга подтверждает эту гипотезу
- Рекомендую посмотреть лекцию Дмитрия Ветрова **Удивительные свойства функции потерь в нейронной сети** (*Surprising properties of loss landscape in overparameterized models*). видео, Презентация
- Автор канала Свидетели Градиента собирает интересные наблюдения и эксперименты про гроккинг.
- Также есть видео с его докладом **Чем не является гроккинг**.

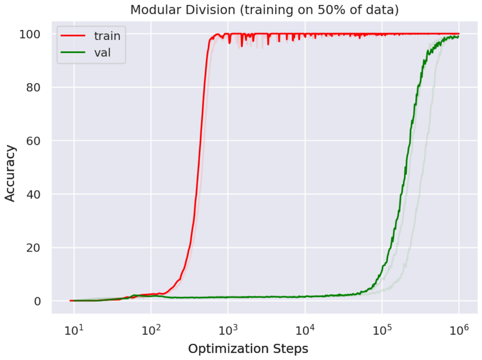


Рис. 10: Гроккинг: Яркий пример обобщения, наступающего намного позже переобучения на алгоритмическом наборе данных.

[9] Power, Alethea, et al. "Grokking: Generalization beyond overfitting on small algorithmic datasets." (2022).

Двойной спуск (Double Descent)

Двойной спуск (Double Descent)[10]

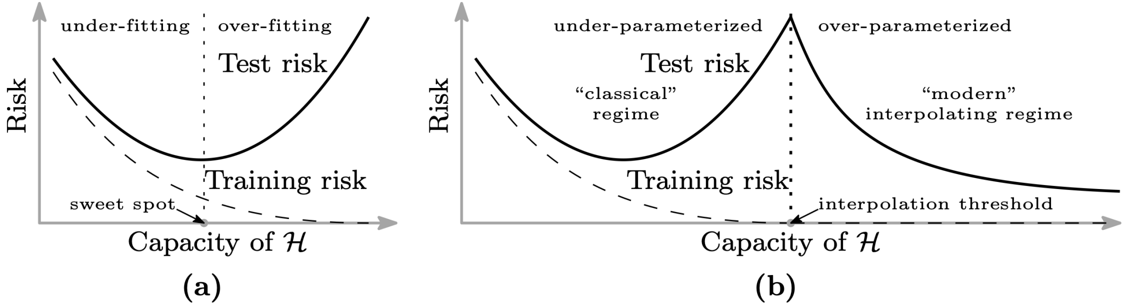


Рис. 11: Кривые риска обучения (пунктирная линия) и тестового риска (сплошная линия). (a) Классическая U-образная кривая риска, возникающая из компромисса смещения и дисперсии (bias-variance trade-off). (b) Кривая риска двойного спуска, которая объединяет U-образную кривую риска (т.е. «классический» режим) с наблюдаемым поведением на моделях высокой размерности (т.е. «современный» интерполяционный режим), разделенных порогом интерполяции. Предикторы справа от порога интерполяции имеют нулевой риск обучения.

[10] Belkin, Mikhail, et al. “Reconciling modern machine-learning practice and the classical bias–variance trade-off.” (2019)

Двойной спуск (Double Descent)