

# Автоматическое дифференцирование

Даня Меркулов

## 1 Автоматическое дифференцирование



@dpiponi@mathstodon.xyz  
@sigfpe



I think the first 40 years or so of automatic differentiation was largely people not using it because they didn't believe such an algorithm could possibly exist.

11:36 PM · Sep 17, 2019



9



26



159



13



Рисунок 1: Когда вы поняли идею



Рисунок 2: Это не autograd

## 1.1 Задача

Предположим, что мы хотим решить следующую задачу:


$$L(w) \rightarrow \min_{w \in \mathbb{R}^d}$$

- Такие задачи обычно возникают в машинном обучении, когда нам нужно найти подходящие параметры  $w$  модели (например, обучить нейронную сеть).
- Существуют разные методы решения этой задачи. Однако, размерность задач сегодня может достигать сотен миллиардов или даже триллионов переменных. Такие задачи очень тяжело решать без знания градиентов, то есть методами нулевого порядка.
- Поэтому было бы полезно уметь вычислять вектор градиента  $\nabla_w L = \left( \frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_d} \right)^T$ .
- Обычно методы первого порядка работают лучше в больших задачах, в то время как методы второго порядка требуют слишком много памяти.

## 1.2 Задача многомерного шкалирования

Предположим, что у нас есть матрица расстояний для  $N$   $d$ -мерных объектов  $D \in \mathbb{R}^{N \times N}$ . Используя эту матрицу, мы хотим восстановить исходные координаты  $W_i \in \mathbb{R}^d$ ,  $i = 1, \dots, N$ .

$$L(W) = \sum_{i,j=1}^N (\|W_i - W_j\|_2^2 - D_{i,j})^2 \rightarrow \min_{W \in \mathbb{R}^{N \times d}}$$

Ссылка на визуализацию , где можно увидеть, что безградиентные методы оптимизации решают эту задачу намного медленнее, особенно в пространствах большой размерности.

## Question

Связано ли это с PCA?

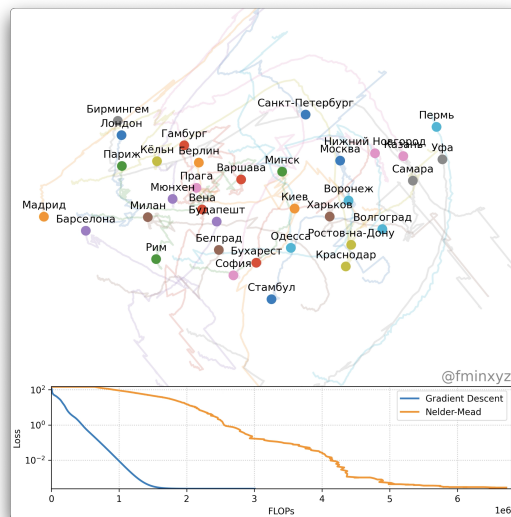


Рисунок 3: [Ссылка на анимацию](#)

### 1.3 Градиентный спуск без градиента

Предположим, что мы хотим решить следующую задачу:

$$L(w) \rightarrow \min_{w \in \mathbb{R}^d}$$

с помощью алгоритма градиентного спуска (GD):

$$w_{k+1} = w_k - \alpha_k \nabla_w L(w_k)$$

Можно ли заменить  $\nabla_w L(w_k)$  используя только информацию нулевого порядка?

Да, но за определенную цену.

Рассмотрим двухточечную оценку градиента<sup>1</sup>  $G$ :

$$G = d \frac{L(w + \varepsilon v) - L(w - \varepsilon v)}{2\varepsilon} v,$$

где  $v$  сферически симметричен.

<sup>1</sup>рекомендуется [хорошая](#) презентация о безградиентных методах

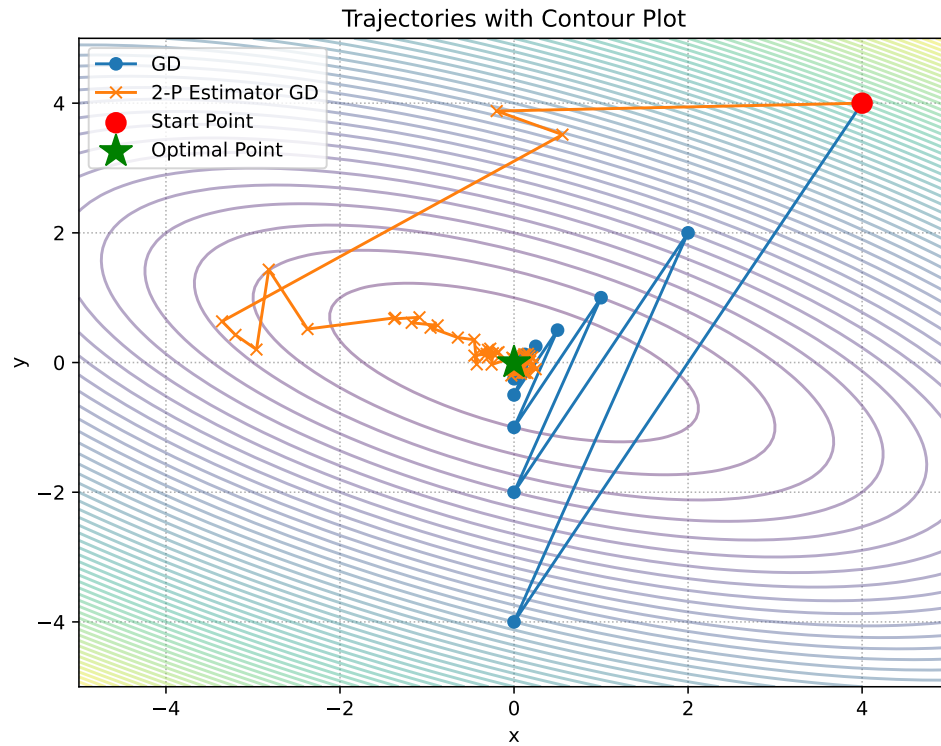


Рисунок 4: “Иллюстрация двухточечной оценки градиентного спуска”

## 1.4 Конечные разности

$$w_{k+1} = w_k - \alpha_k G$$

Также рассмотрим идею конечных разностей:

$$G = \sum_{i=1}^d \frac{L(w + \varepsilon e_i) - L(w - \varepsilon e_i)}{2\varepsilon} e_i$$

[Открыть в Colab](#)

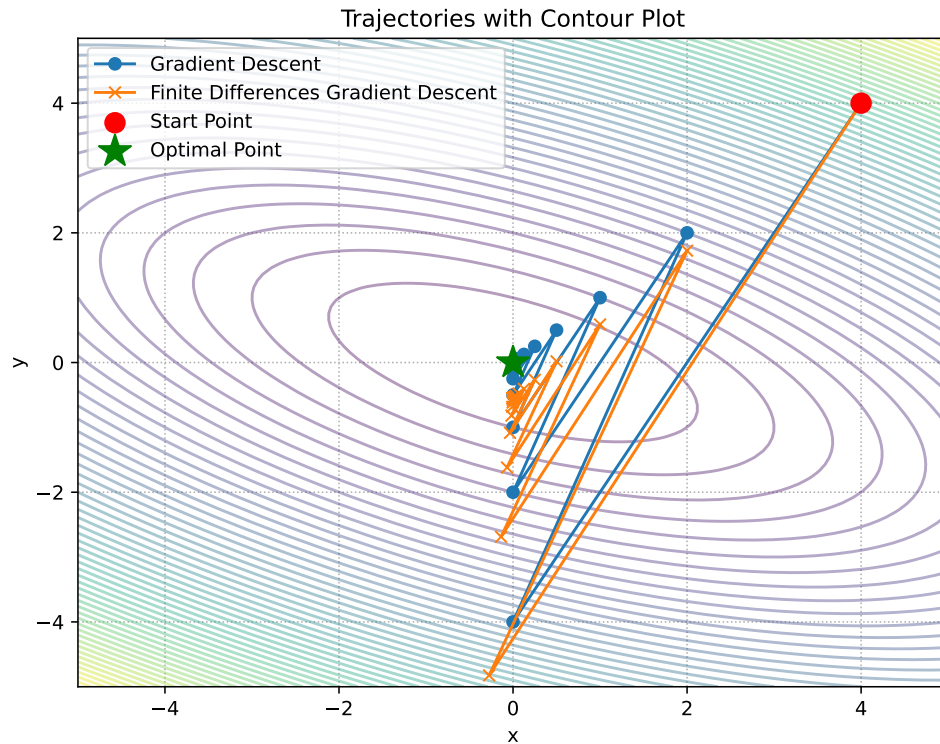


Рисунок 5: “Иллюстрация работы метода оценки градиента с помощью метода конечных разностей”

## 1.5 Проклятие размерности для методов нулевого порядка<sup>2</sup>

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

$$\text{GD: } x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

$$\text{Zero order GD: } x_{k+1} = x_k - \alpha_k G,$$

где  $G$  - оценка градиента 2-точечная или многоточечная.

	$f(x)$ - гладкая	$f(x)$ - гладкая и выпуклая	$f(x)$ - гладкая и сильно выпуклая
GD	$\ \nabla f(x_k)\ ^2 \approx \mathcal{O}\left(\frac{1}{k}\right)$	$f(x_k) - f^* \approx \mathcal{O}\left(\frac{1}{k}\right)$	$\ x_k - x^*\ ^2 \approx \mathcal{O}\left(\left(1 - \frac{\mu}{L}\right)^k\right)$
GD нулевого порядка	$\ \nabla f(x_k)\ ^2 \approx \mathcal{O}\left(\frac{n}{k}\right)$	$f(x_k) - f^* \approx \mathcal{O}\left(\frac{n}{k}\right)$	$\ x_k - x^*\ ^2 \approx \mathcal{O}\left(\left(1 - \frac{\mu}{nL}\right)^k\right)$

Для 2-точечных оценок, мы не можем сделать зависимость лучше, чем от  $\sqrt{n}$  !

<sup>2</sup>Оптимальные скорости для нулевого порядка выпуклой оптимизации: сила двух оценок функции

## 1.6 Конечные разности

Наивный подход к получению приближительных значений градиентов - это подход **конечных разностей**. Для каждой координаты, можно вычислить приближенное значение частной производной:

$$\frac{\partial L}{\partial w_k}(w) \approx \frac{L(w + \varepsilon e_k) - L(w)}{\varepsilon}, \quad e_k = (0, \dots, 1, \dots, 0)$$

### i Question

Если время, необходимое для одного вычисления  $L(w)$  равно  $T$ , то какое время необходимо для вычисления  $\nabla_w L$  с этим подходом?

**Ответ**  $2dT$ , что очень долго для больших задач. Кроме того, этот метод нестабилен, что означает, что нам придется выбирать между точностью и стабильностью.

### Теорема

Существует алгоритм для вычисления  $\nabla_w L$  за  $\mathcal{O}(T)$ .<sup>3</sup>

## 1.7 Прямой режим автоматического дифференцирования

Чтобы глубже понять идею автоматического дифференцирования, рассмотрим простую функцию для вычисления производных:

$$L(w_1, w_2) = w_2 \log w_1 + \sqrt{w_2 \log w_1}$$

Давайте нарисуем *вычислительный граф* этой функции:

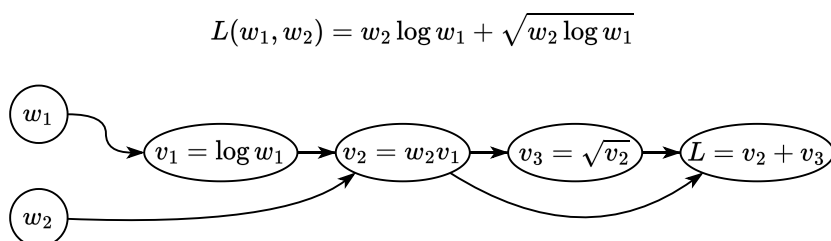


Рисунок 6: Иллюстрация вычислительного графа для функции  $L(w_1, w_2)$

Давайте пойдем от начала графа к концу и вычислим производную  $\frac{\partial L}{\partial w_1}$ .

<sup>3</sup>Linnainmaa S. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, 1970.

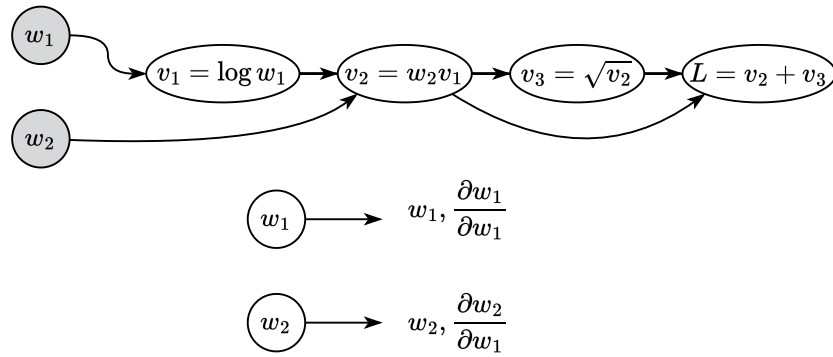


Рисунок 7: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$w_1 = w_1, w_2 = w_2$$

### Производная

$$\frac{\partial w_1}{\partial w_1} = 1, \frac{\partial w_2}{\partial w_1} = 0$$

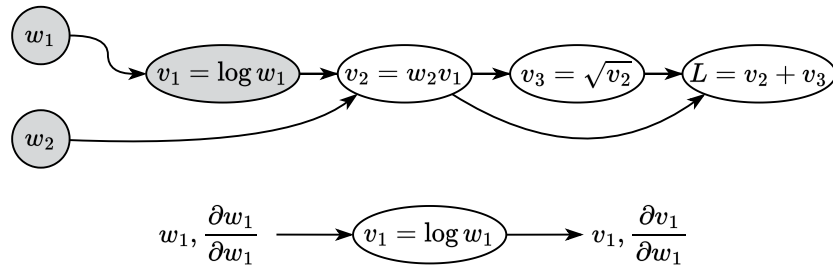


Рисунок 8: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_1 = \log w_1$$

### Производная

$$\frac{\partial v_1}{\partial w_1} = \frac{\partial v_1}{\partial w_1} \frac{\partial w_1}{\partial w_1} = \frac{1}{w_1} 1$$

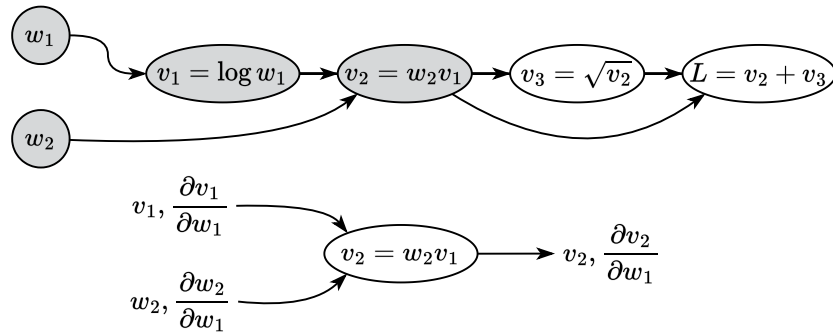


Рисунок 9: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_2 = w_2 v_1$$

### Производная

$$\frac{\partial v_2}{\partial w_1} = \frac{\partial v_2}{\partial v_1} \frac{\partial v_1}{\partial w_1} + \frac{\partial v_2}{\partial w_2} \frac{\partial w_2}{\partial w_1} = w_2 \frac{\partial v_1}{\partial w_1} + v_1 \frac{\partial w_2}{\partial w_1}$$

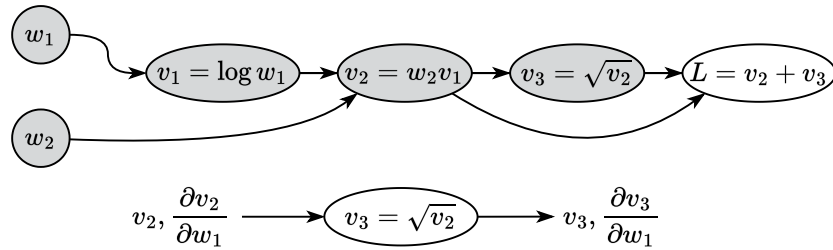


Рисунок 10: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_3 = \sqrt{v_2}$$

### Производная

$$\frac{\partial v_3}{\partial w_1} = \frac{\partial v_3}{\partial v_2} \frac{\partial v_2}{\partial w_1} = \frac{1}{2\sqrt{v_2}} \frac{\partial v_2}{\partial w_1}$$

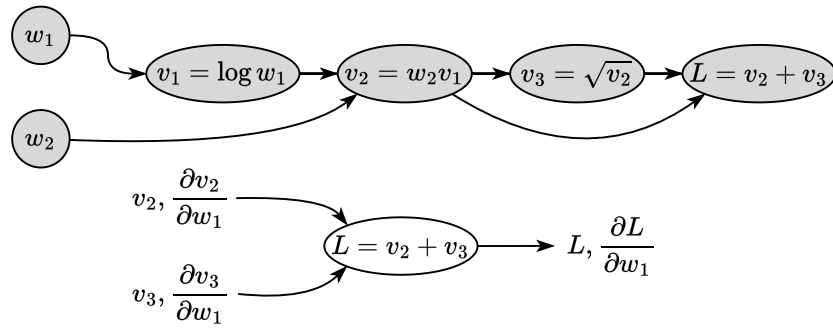


Рисунок 11: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$L = v_2 + v_3$$

### Производная

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial v_2} \frac{\partial v_2}{\partial w_1} + \frac{\partial L}{\partial v_3} \frac{\partial v_3}{\partial w_1} = 1 \frac{\partial v_2}{\partial w_1} + 1 \frac{\partial v_3}{\partial w_1}$$

Сделайте аналогичные вычисления для  $\frac{\partial L}{\partial w_2}$

$$L(w_1, w_2) = w_2 \log w_1 + \sqrt{w_2 \log w_1}$$

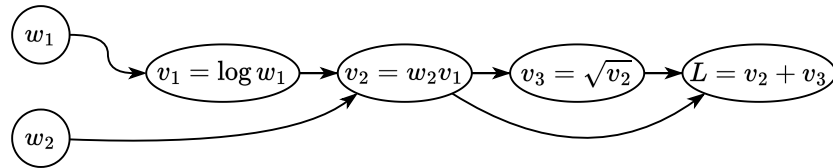


Рисунок 12: Иллюстрация вычислительного графа для функции  $L(w_1, w_2)$

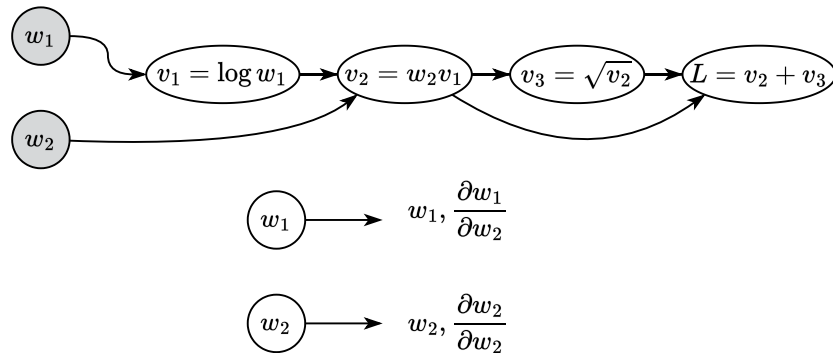


Рисунок 13: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$w_1 = w_1, w_2 = w_2$$

### Производная

$$\frac{\partial w_1}{\partial w_2} = 0, \frac{\partial w_2}{\partial w_2} = 1$$

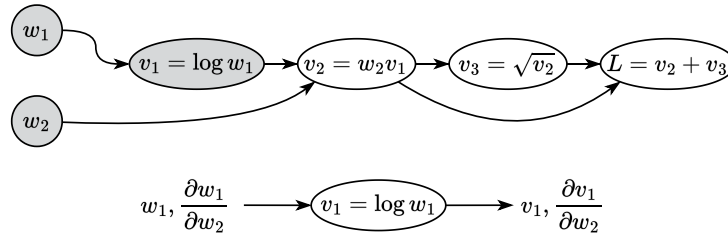


Рисунок 14: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_1 = \log w_1$$

### Производная

$$\frac{\partial v_1}{\partial w_2} = \frac{\partial v_1}{\partial w_1} \frac{\partial w_1}{\partial w_2} = \frac{1}{w_1} \cdot 0$$

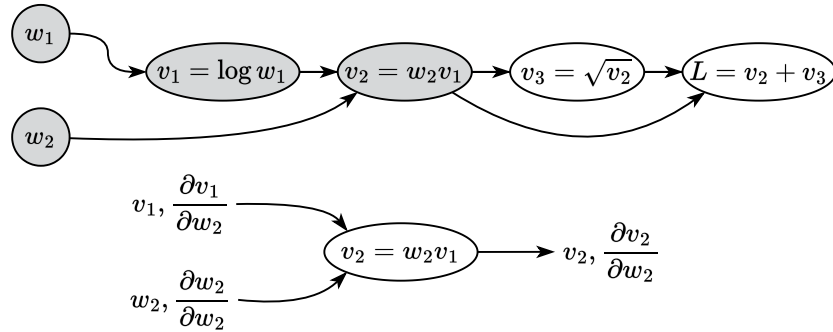


Рисунок 15: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_2 = w_2 v_1$$

### Производная

$$\frac{\partial v_2}{\partial w_2} = \frac{\partial v_2}{\partial v_1} \frac{\partial v_1}{\partial w_2} + \frac{\partial v_2}{\partial w_2} \frac{\partial w_2}{\partial w_2} = w_2 \frac{\partial v_1}{\partial w_2} + v_1 \frac{\partial w_2}{\partial w_2}$$

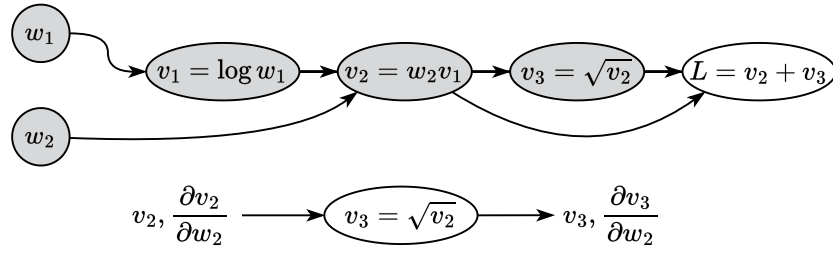


Рисунок 16: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$v_3 = \sqrt{v_2}$$

### Производная

$$\frac{\partial v_3}{\partial w_2} = \frac{\partial v_3}{\partial v_2} \frac{\partial v_2}{\partial w_2} = \frac{1}{2\sqrt{v_2}} \frac{\partial v_2}{\partial w_2}$$

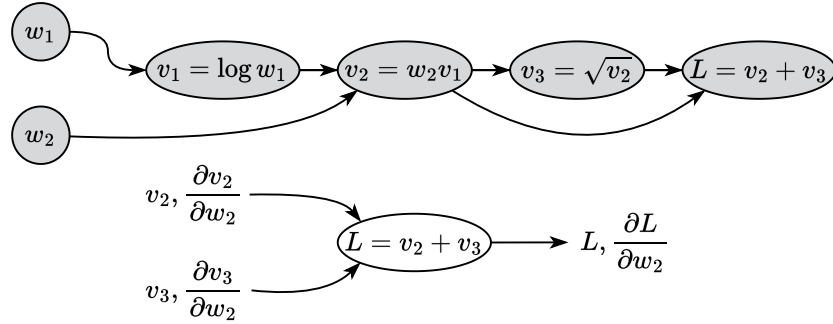


Рисунок 17: Иллюстрация прямого режима автоматического дифференцирования

### Функция

$$L = v_2 + v_3$$

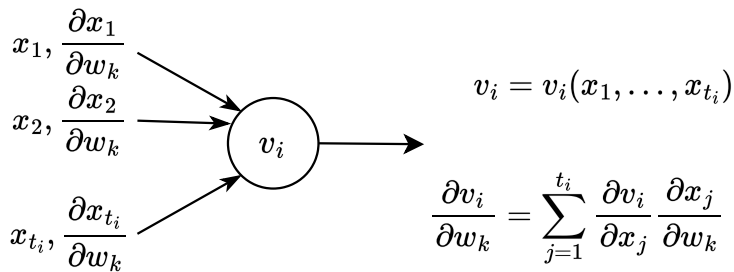
### Производная

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial v_2} \frac{\partial v_2}{\partial w_2} + \frac{\partial L}{\partial v_3} \frac{\partial v_3}{\partial w_2} = 1 \frac{\partial v_2}{\partial w_2} + 1 \frac{\partial v_3}{\partial w_2}$$

## 1.8 Алгоритм прямого режима автоматического дифференцирования

Предположим, что у нас есть вычислительный граф  $v_i, i \in [1; N]$ . Наша цель - вычислить производную выхода этого графа по некоторой входной переменной  $w_k$ , т.е.  $\frac{\partial v_N}{\partial w_k}$ . Эта идея предполагает распространение градиента по входной переменной от начала к концу, поэтому мы можем ввести обозначение:

$$\overline{v}_i = \frac{\partial v_i}{\partial w_k}$$



- Для  $i = 1, \dots, N$ :
  - Вычислить  $v_i$  как функцию его предков  $x_1, \dots, x_{t_i}$ :

$$v_i = v_i(x_1, \dots, x_{t_i})$$

- Вычислить производную  $\overline{v}_i$  используя формулу производной сложной функции:

$$\overline{v}_i = \sum_{j=1}^{t_i} \frac{\partial v_i}{\partial x_j} \frac{\partial x_j}{\partial w_k}$$

Обратите внимание, что этот подход не требует хранения всех промежуточных вычислений, но можно видеть, что для вычисления производной  $\frac{\partial L}{\partial w_k}$  нам нужно  $\mathcal{O}(T)$  операций. Это означает, что для всего градиента, нам нужно  $d\mathcal{O}(T)$  операций, что то же самое, что и для конечных разностей, но теперь у нас нет проблем со стабильностью или неточностями (формулы выше точны).



## 1.9 Обратный режим автоматического дифференцирования

Мы рассмотрим ту же функцию с вычислительным графом:

$$L(w_1, w_2) = w_2 \log w_1 + \sqrt{w_2 \log w_1}$$

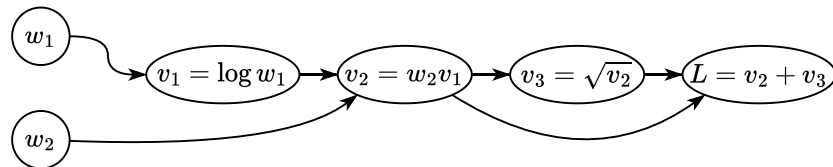


Рисунок 18: Иллюстрация вычислительного графа для функции  $L(w_1, w_2)$

Предположим, что у нас есть некоторые значения параметров  $w_1, w_2$  и мы уже выполнили прямой проход (т.е. вычисление значений всех промежуточных узлов вычислительного графа). Предположим также, что мы как-то сохранили все промежуточные значения  $v_i$ . Давайте пойдём от конца графа к началу и вычислим производные  $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}$ :

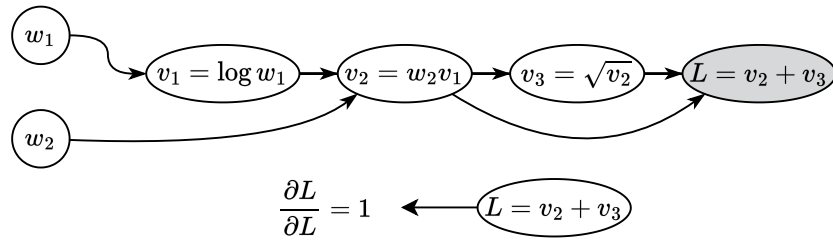


Рисунок 19: Иллюстрация обратного режима автоматического дифференцирования

## Производные

$$\frac{\partial L}{\partial L} = 1$$

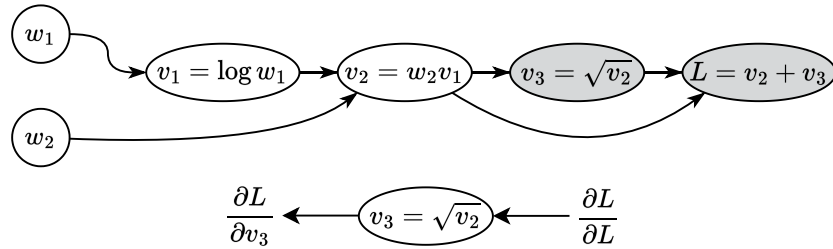


Рисунок 20: Иллюстрация обратного режима автоматического дифференцирования

## Производные

$$\frac{\partial L}{\partial v_3} = \frac{\partial L}{\partial L} \frac{\partial L}{\partial v_3} = \frac{\partial L}{\partial L} 1$$

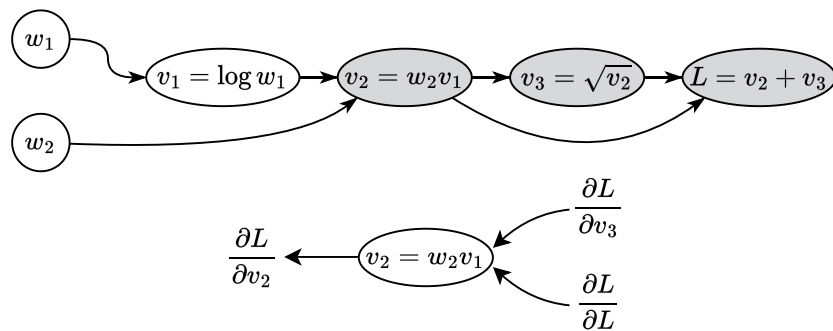


Рисунок 21: Иллюстрация обратного режима автоматического дифференцирования

## Производные

$$\frac{\partial L}{\partial v_2} = \frac{\partial L}{\partial v_3} \frac{\partial v_3}{\partial v_2} + \frac{\partial L}{\partial L} \frac{\partial L}{\partial v_2} = \frac{\partial L}{\partial v_3} \frac{1}{2\sqrt{v_2}} + \frac{\partial L}{\partial L} 1$$

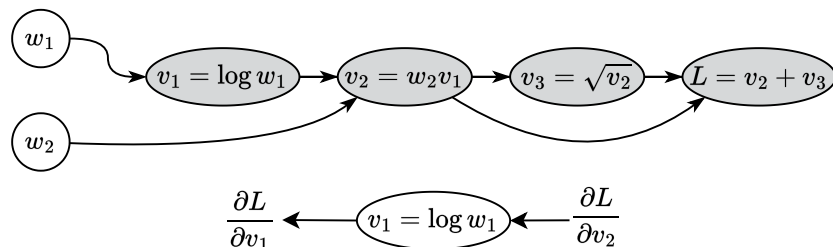


Рисунок 22: Иллюстрация обратного режима автоматического дифференцирования

## Производные

$$\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial v_2} \frac{\partial v_2}{\partial v_1} = \frac{\partial L}{\partial v_2} w_2$$

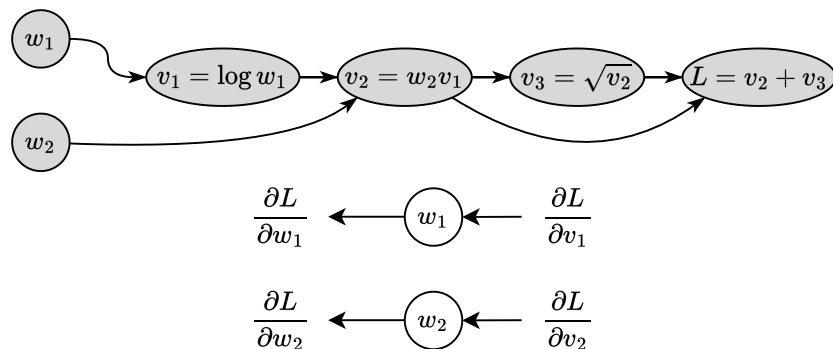


Рисунок 23: Иллюстрация обратного режима автоматического дифференцирования

## Производные

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial v_1} \frac{\partial v_1}{\partial w_1} = \frac{\partial L}{\partial v_1} \frac{1}{w_1} \quad \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial v_2} \frac{\partial v_2}{\partial w_2} = \frac{\partial L}{\partial v_1} v_1$$

### 1.10 Обратный режим автоматического дифференцирования

#### i Question

Обратите внимание, что для того же количества вычислений, что и в прямом режиме, мы получаем полный вектор градиента  $\nabla_w L$ . Какова стоимость ускорения?

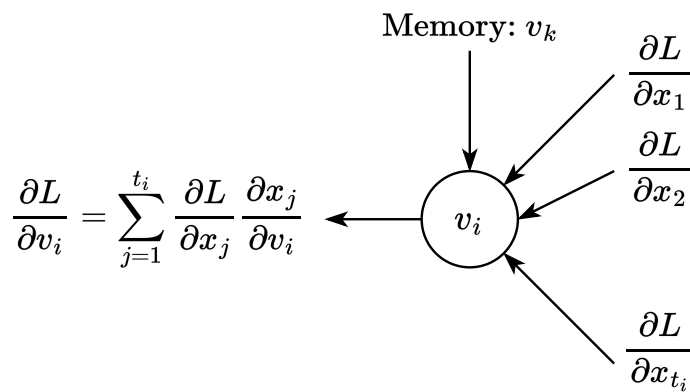
**Ответ** Обратите внимание, что для использования обратного режима AD вам нужно хранить все промежуточные вычисления из прямого прохода. Эта проблема может быть частично решена с

помощью чекпоинтинга, при котором мы сохраняем только часть промежуточных значений, а остальные пересчитываем заново по мере необходимости. Это позволяет значительно уменьшить объём требуемой памяти при обучении больших моделей машинного обучения.

### 1.11 Алгоритм обратного режима автоматического дифференцирования

Предположим, что у нас есть вычислительный граф  $v_i, i \in [1; N]$ . Наша цель - вычислить производную выхода этого графа по всем входным переменным  $w$ , т.е.  $\nabla_w v_N = \left( \frac{\partial v_N}{\partial w_1}, \dots, \frac{\partial v_N}{\partial w_d} \right)^T$ . Эта идея предполагает распространение градиента функции по промежуточным переменным от конца к началу, поэтому мы можем ввести обозначение:

$$\bar{v}_i = \frac{\partial L}{\partial v_i} = \frac{\partial v_N}{\partial v_i}$$



#### • ПРЯМОЙ ПРОХОД

Для  $i = 1, \dots, N$ :

- Вычислить и сохранить значения  $v_i$  как функцию его предков

#### • ОБРАТНЫЙ ПРОХОД

Для  $i = N, \dots, 1$ :

- Вычислить производную  $\bar{v}_i$  используя формулу производной сложной функции и информацию от всех потомков (выходов):

$$\bar{v}_i = \frac{\partial L}{\partial v_i} = \sum_{j=1}^{t_i} \frac{\partial L}{\partial x_j} \frac{\partial x_j}{\partial v_i}$$

## Choose your fighter

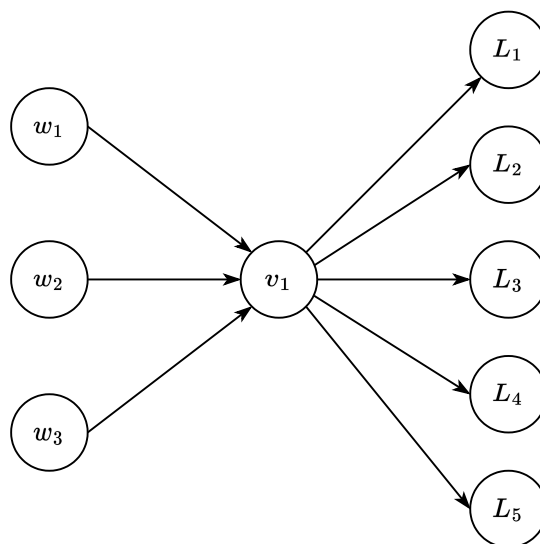


Рисунок 24: Какой режим вы бы выбрали для вычисления градиентов?

**i** Question

Какой из режимов AD вы бы выбрали (прямой/обратный) для следующего вычислительного графа арифметических операций? Предположим, что вам нужно вычислить якобиан  $J = \left\{ \frac{\partial L_i}{\partial w_j} \right\}_{i,j}$

**Ответ** Обратите внимание, что время вычислений в обратном режиме пропорционально количеству выходов, тогда как время работы прямого режима пропорционально количеству входов. Поэтому было бы хорошей идеей рассмотреть прямой режим AD.

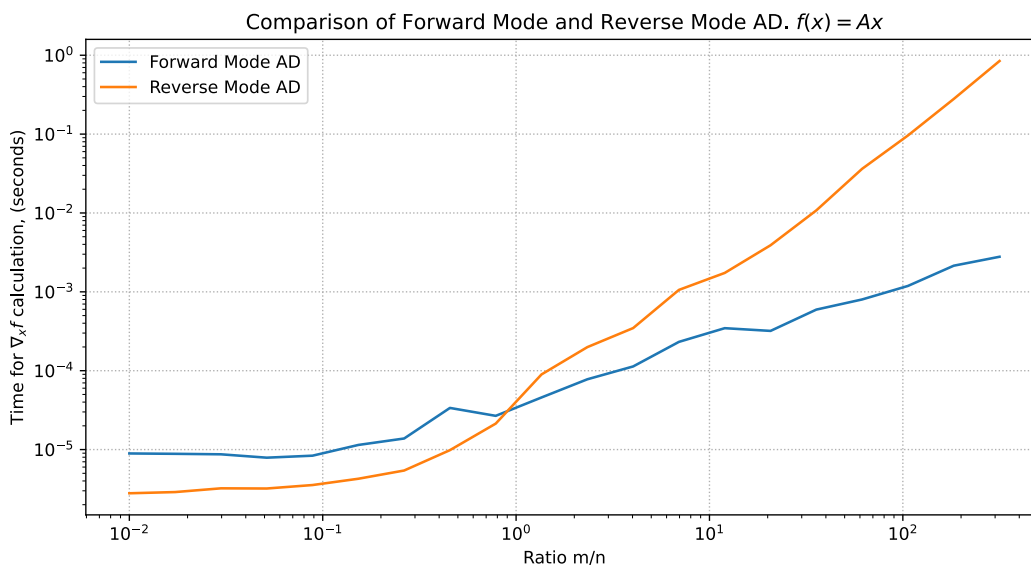


Рисунок 25: 🍀 График иллюстрирует идею выбора между режимами автоматического дифференцирования. Размерность входа  $n = 100$  фиксирована, измерено время вычисления якобиана в зависимости от соотношения размерностей выхода и входа для разных размерностей выхода  $m$ .

### Choose your fighter

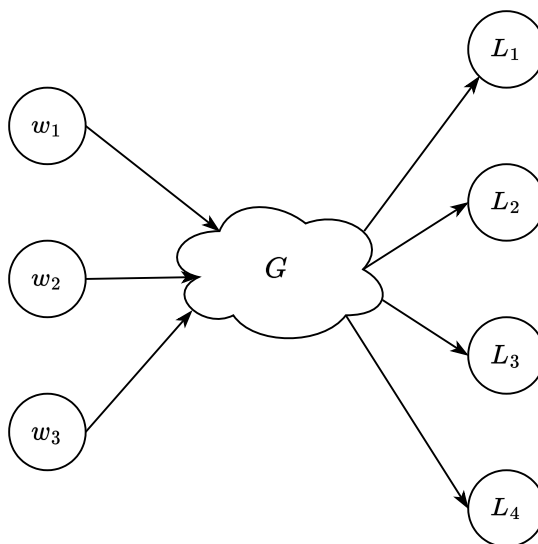


Рисунок 26: Какой режим вы бы выбрали для вычисления градиентов?

#### Question

Какой из режимов AD вы бы выбрали (прямой/обратный) для следующего вычислительного графа арифметических операций? Предположим, что вам нужно вычислить якобиан  $J = \left\{ \frac{\partial L_i}{\partial w_j} \right\}_{i,j}$ .

Обратите внимание, что  $G$  - это произвольный вычислительный граф

**Ответ** В общем случае невозможно ответить без некоторого знания о конкретной структуре графа  $G$ . Следует отметить, что существуют продвинутое подходы, смешивающие прямой и обратный режим AD в зависимости от конкретной структуры графа  $G$ .

## 1.12 Архитектура прямого распространения

### ПРЯМОЙ ПРОХОД

- $v_0 = x$  на вход обычно подаётся батч данных  $x$
- Для  $k = 1, \dots, t-1, t$ :
  - $v_k = \sigma(v_{k-1} w_k)$ . Обратите внимание, что на практике, данные имеют размерность  $x \in \mathbb{R}^{b \times d}$ , где  $b$  - размер батча (для одного объекта из выборки  $b = 1$ ). В то время как матрица весов  $w_k$   $k$  слоя имеет размер  $n_{k-1} \times n_k$ , где  $n_k$  - размер внутреннего представления данных.
- $L = L(v_t)$  - вычислить функцию потерь.

### ОБРАТНЫЙ ПРОХОД

- $v_{t+1} = L, \frac{\partial L}{\partial L} = 1$
- Для  $k = t, t-1, \dots, 1$ :
  - $\frac{\partial L}{\partial v_k} = \frac{\partial L}{\partial v_{k+1}} \frac{\partial v_{k+1}}{\partial v_k}$   
 $b \times n_k \quad b \times n_{k+1} \quad n_{k+1} \times n_k$
  - $\frac{\partial L}{\partial w_k} = \frac{\partial L}{\partial v_{k+1}} \cdot \frac{\partial v_{k+1}}{\partial w_k}$   
 $b \times n_{k-1} \cdot n_k \quad b \times n_{k+1} \quad n_{k+1} \times n_{k-1} \cdot n_k$

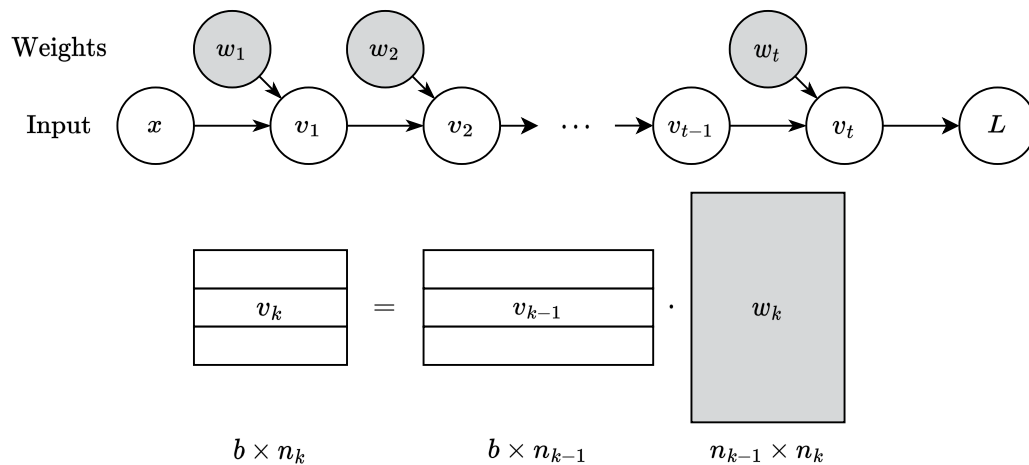


Рисунок 27: Архитектура прямого распространения нейронной сети

### 1.13 Произведение Гессиана на вектор без вычисления самого Гессиана

Когда вам нужна некоторая информация о кривизне функции, обычно вам нужно работать с гессианом. Однако, это трудно делать, когда размерность задачи велика. Для скалярной функции  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , гессиан в точке  $x \in \mathbb{R}^n$  записывается как  $\nabla^2 f(x)$ . Тогда произведение вектора на гессиан можно записать как

$$v \mapsto \nabla^2 f(x) \cdot v$$

для любого вектора  $v \in \mathbb{R}^n$ . Мы можем использовать тождество

$$\nabla^2 f(x)v = \nabla[x \mapsto \nabla f(x)^T \cdot v] = \nabla g(x),$$

где  $g(x) = \nabla f(x)^T \cdot v$  - новая функция, которая скалярно умножает градиент  $f$  в  $x$  на вектор  $v$ .

```
import jax.numpy as jnp

def hvp(f, x, v):
    return grad(lambda x: jnp.vdot(grad(f)(x), v))(x)
```

### 1.14 Динамика обучения нейронной сети через спектр Гессиана и hvp <sup>4</sup>

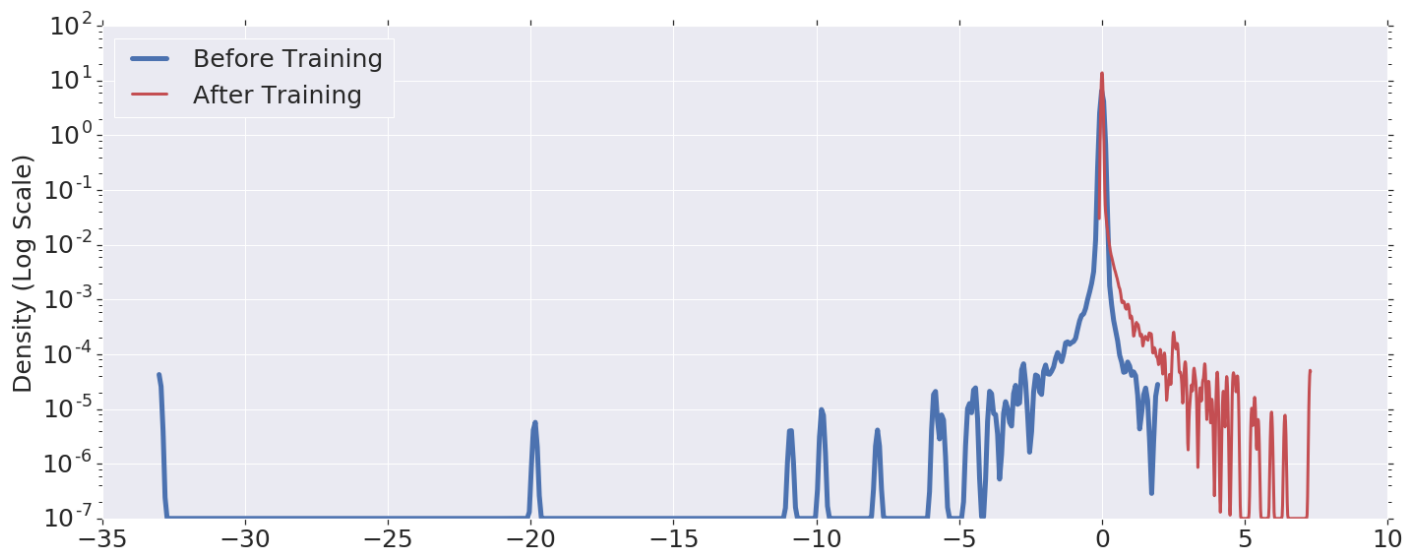


Рисунок 28: Большие по модулю отрицательные собственные значения гессиана исчезли после обучения ResNet-32

### 1.15 Идея Хадчинсона для оценки следа матрицы <sup>5</sup>

Метод Хатчинсона позволяет оценить след гессиана с помощью операций вычисления умножения гессиана на произвольный вектор:

<sup>4</sup>Некоторые исследования в оптимизации нейронных сетей через спектр собственных значений Гессиана

<sup>5</sup>A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines - M.F. Hutchinson, 1990

Пусть  $X \in \mathbb{R}^{d \times d}$  и  $v \in \mathbb{R}^d$  - случайный вектор такой, что  $\mathbb{E}[vv^T] = I$ . Тогда,

$$\text{Tr}(X) = \mathbb{E}[v^T X v] = \frac{1}{V} \sum_{i=1}^V v_i^T X v_i.$$

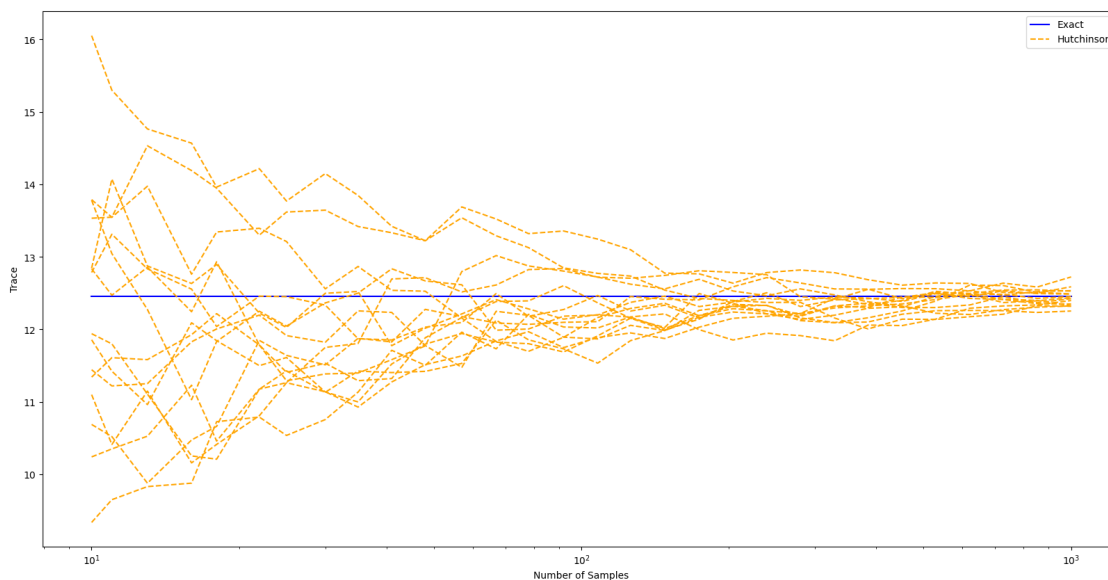


Рисунок 29: [Источник](#)

## 1.16 Чекпоинтинг

Анимация вышеуказанных подходов

Пример использования контрольных точек градиента

В качестве примера рассмотрим обучение **GPT-2**<sup>6</sup>:

- Активации в простом режиме могут занимать гораздо больше памяти: для последовательности длиной 1K и размера батча 32, 60 GB нужно для хранения всех промежуточных активаций.
- Чекпоинтинг может снизить потребление до 8 GB, пересчитывая их (33% дополнительных вычислений)

## 1.17 Чем автоматическое дифференцирование (AD) не является:

- AD не является методом конечных разностей
- AD не является символьным вычислением производных
- AD не является только правилом вычисления производной сложной функции
- AD (обратный режим) является времяэффективным и численно стабильным
- AD (обратный режим) не является эффективным по памяти (нужно хранить все промежуточные вычисления из прямого прохода)

<sup>6</sup>[ZeRO: Memory Optimizations Toward Training Trillion Parameter Models](#)

# DIFFERENTIATION

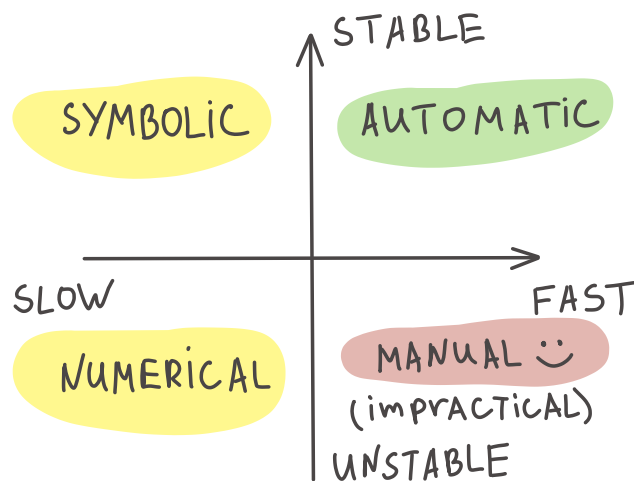


Рисунок 30: Различные подходы для взятия производных

## 1.18 Дополнительные материалы

- Рекомендую прочитать официальную книгу по Jax Autodiff. [Open In Colab](#) ♣
- Распространение градиента через линейные наименьшие квадраты [семинар]
- Распространение градиента через SVD [семинар]
- Контрольные точки активаций [семинар]

## 2 Итоги

### 2.1 Определения

1. Формула для приближенного вычисления производной функции  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$  по  $k$ -ой координате с помощью метода конечных разностей.
2. Пусть  $f = f(x_1(t), \dots, x_n(t))$ . Формула для вычисления  $\frac{\partial f}{\partial t}$  через  $\frac{\partial x_i}{\partial t}$  (Forward chain rule).
3. Пусть  $L$  - функция, возвращающая скаляр, а  $v_k$  - функция, возвращающая вектор  $x \in \mathbb{R}^t$ . Формула для вычисления  $\frac{\partial L}{\partial v_k}$  через  $\frac{\partial L}{\partial x_i}$  (Backward chain rule).
4. Идея Хатчинсона для оценки следа матрицы с помощью matvec операций.

### 2.2 Теоремы

1. Автоматическое дифференцирование. Вычислительный граф. Forward/ Backward mode (в этом вопросе нет доказательств, но необходимо подробно описать алгоритмы).

## 3 Задачи

### 3.1 Задача 1

#### i Question

Какой из режимов AD вы бы выбрали (прямой/обратный) для следующего вычислительного графа арифметических операций?

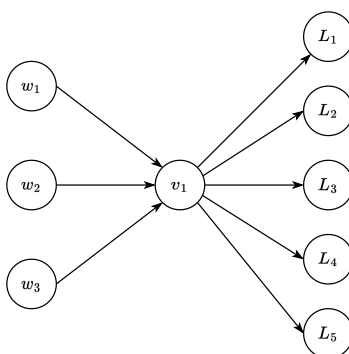


Рисунок 31: Какой режим вы бы выбрали для вычисления градиентов?

### 3.2 Задача 2

Предположим, у нас есть обратимая матрица  $A$  и вектор  $b$ , вектор  $x$  является решением системы линейных уравнений  $Ax = b$ , то есть можно записать аналитическое решение  $x = A^{-1}b$ .

#### i Question

Найдите производные  $\frac{\partial L}{\partial A}$ ,  $\frac{\partial L}{\partial b}$ .

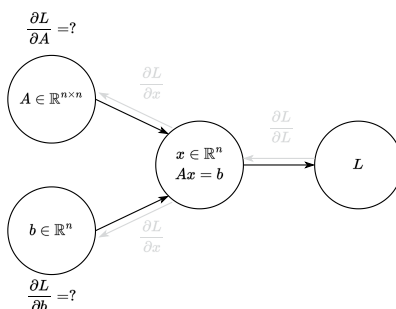


Рисунок 32:  $x$  может быть найден как решение линейной системы

### 3.3 Задача 3

Предположим, у нас есть прямоугольная матрица  $W \in \mathbb{R}^{m \times n}$ , которая имеет сингулярное разложение:

$$W = U\Sigma V^T, \quad U^T U = I, \quad V^T V = I, \\ \Sigma = \text{diag}(\sigma_1, \dots, \sigma_{\min(m,n)})$$

Регуляризатор  $R(W) = \text{tr}(\Sigma)$  в любой функции потерь стимулирует низкоранговые решения.

#### Question

Найдите производную  $\frac{\partial R}{\partial W}$ .

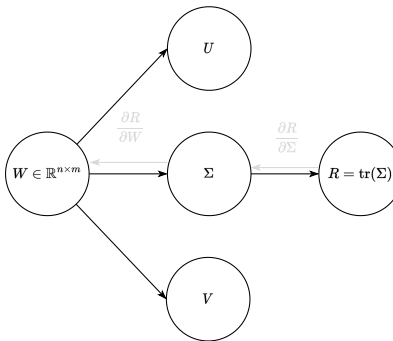


Рисунок 33: Вычислительный граф для сингулярного регуляризатора

## 4 Задачи на дом

1. **Бенчмаркинг вычисления произведения Гессиана на вектор (HVP) в нейронной сети через JAX [22 балла]**

Вам дана простая нейронная сеть (MLP с несколькими скрытыми слоями с нелинейностью, такой как GELU). Параметры модели определяются весами ее слоев. Ваша задача состоит в том, чтобы сравнить различные подходы для вычисления произведения Гессиана на вектор (HVP) по отношению к функции потерь модели и изучить, как время вычисления масштабируется с ростом модели.

**Определение модели и функции потерь:** [2/22 балла] Вот код для определения модели и функции потерь. Напишите метод `get_params()`, который возвращает сглаженный вектор всех весов модели.

```
import jax
import jax.numpy as jnp
import time
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm.auto import tqdm
from jax.nn import gelu

#      MLP
class MLP:
    def __init__(self, key, layer_sizes):
        self.layer_sizes = layer_sizes
        keys = jax.random.split(key, len(layer_sizes) - 1)
        self.weights = [
            jax.random.normal(k, (layer_sizes[i], layer_sizes[i + 1]))
            for i, k in enumerate(keys)
        ]

    def forward(self, x):
        for w in self.weights[:-1]:
            x = gelu(jnp.dot(x, w))
        return jnp.dot(x, self.weights[-1])

    def get_params(self):
        ### YOUR CODE HERE ###
        return None
```

**Гессиан и HVP: [2/22 балла]** Напишите функцию

```
#
def calculate_hessian(model, params):
    def loss_fn(p):
        x = jnp.ones((1, model.layer_sizes[0])) #
        return jnp.sum(model.forward(x))

    ### YOUR CODE HERE ###
    #hessian_fn =
    return hessian_fn(params)
```

которая вычисляет полный Гессиан  $H$  функции потерь по отношению к параметрам модели с использованием автоматического дифференцирования JAX.

**Naïve HVP через полный Гессиан: [2/22 балла]** Напишите функцию `naive_hvp(hessian, vector)`, которая, учитывая предварительно вычисленный Гессиан  $H$  и вектор  $v$  (той же формы, что и параметры), вычисляет произведение Гессиана на вектор с использованием прямого матрично-векторного умножения.

**Эффективное HVP с использованием Autograd: [4/22 балла]** Напишите функцию python `def hvp(f, x, v):` `return jax.grad(lambda x: jnp.vdot(jax.grad(f)(x), v))(x)` которая непосредственно вычисляет HVP без явного формирования полного Гессиана. Это использует возможности обратного режима дифференцирования JAX.

**Эксперимент по времени:** Рассмотрим семейство моделей с увеличивающимся количеством скрытых слоев.

```
ns = np.linspace(50, 1000, 15, dtype=int) # The number of hidden layers
num_runs = 10 # The number of runs for averaging
```

Для каждой конфигурации модели:

- Сгенерируйте модель и извлеките ее вектор параметров.
- Сгенерируйте случайный вектор  $v$  той же размерности, что и параметры.
- Измерьте (не забудьте использовать `.block_until_ready()` для обеспечения точного измерения времени и правильной синхронизации) следующее:
  1. **Общее время (полный Гессиан + Naïve HVP):** Общее время, необходимое для вычисления полного Гессиана и затем выполнения матрично-векторного умножения.
  2. **Время Naïve HVP (без вычисления Гессиана):** Время, необходимое для выполнения матрично-векторного умножения, учитывая предварительно вычисленный Гессиан.
  3. **Время эффективного HVP:** Время, необходимое для вычисления HVP с использованием функции на основе autograd.
- Повторите каждое измерение времени для фиксированного количества запусков (например, 10 запусков) и запишите как среднее, так и стандартное отклонение времени вычисления.

**Визуализация и анализ:** [12/22 балла]

- Постройте график результатов времени для трех методов на одном графике. Для каждого метода отобразите ошибки, соответствующие стандартному отклонению по запускам.
  - Четко обозначьте оси (например, “Количество слоев” против “Время вычисления (секунды)”) и включите легенду, указывающую, какая кривая соответствует какому методу.
  - Анализируйте масштабирование. Попробуйте аналитически вывести масштабирование методов и сравнить его с экспериментальными результатами.
2. [15 баллов] Мы можем использовать автоматическое дифференцирование не только для вычисления необходимых градиентов, но и для настройки гиперпараметров алгоритма, таких как скорость обучения в градиентном спуске (с градиентным спуском ☺). Предположим, у нас есть следующая функция  $f(x) = \frac{1}{2}\|x\|^2$ , выберите случайную точку  $x_0 \in \mathbb{B}^{1000} = \{0 \leq x_i \leq 1 \mid \forall i\}$ . Рассмотрим 10 шагов градиентного спуска, начиная с точки  $x_0$ :

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

Ваша цель в этой задаче состоит в том, чтобы написать функцию, которая принимает 10 скалярных значений  $\alpha_i$  и возвращает результат градиентного спуска на функции  $L = f(x_{10})$ . И оптимизируйте эту функцию с помощью градиентного спуска на  $\alpha \in \mathbb{R}^{10}$ . Предположим, что каждый из 10 компонентов  $\alpha$  равномерно распределен на  $[0; 0.1]$ .

$$\alpha_{k+1} = \alpha_k - \beta \frac{\partial L}{\partial \alpha}$$

Выберите любую константу  $\beta$  и количество шагов, которое вам нужно. Опишите полученные результаты. Как вы поймете, что полученный график ( $\alpha \in \mathbb{R}^{10}$ ) стал лучше, чем был в начале? Как вы проверите локальную оптимальность этой задачи численно?