Введение

Игорь Удовиченко, @igortao

Егор Швецов

Игорь Удовиченко

Виктор Москворецкий

Максим Желнин

Дмитрий Осин

>>> Что такое эффективный DL?

Эффективность —
способность решить поставленную задачу,
используя минимум ресурсов

Какие бывают ресурсы?

- время и деньги
  - время обучения/инференса, затраты на электроэнергию/аренду вычислительных мощностей, время разработки
  - FLOPS, используемая память, latency, throughput, overhead...

Мы будем рассматривать именно низкоуровневую эффективность

FLOPS — число операций с плавающей запятой в секунду

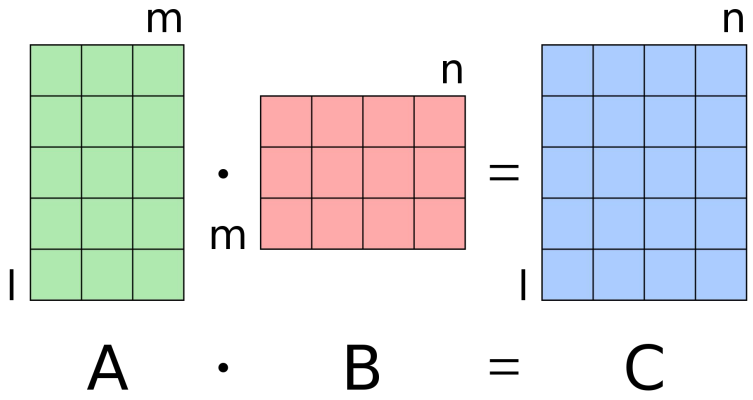
- Дробные числа в компьютере обычно представляются в формате чисел с плавающей запятой, например 2,99792458 * 10^8.
- Арифметические операции с такими числами в железе реализованы нетривиально, и для них разработаны свои модули, которые сложнее, чем для целых чисел.
- Поэтому операции с плавающей запятой являются самостоятельной мерой сложности.

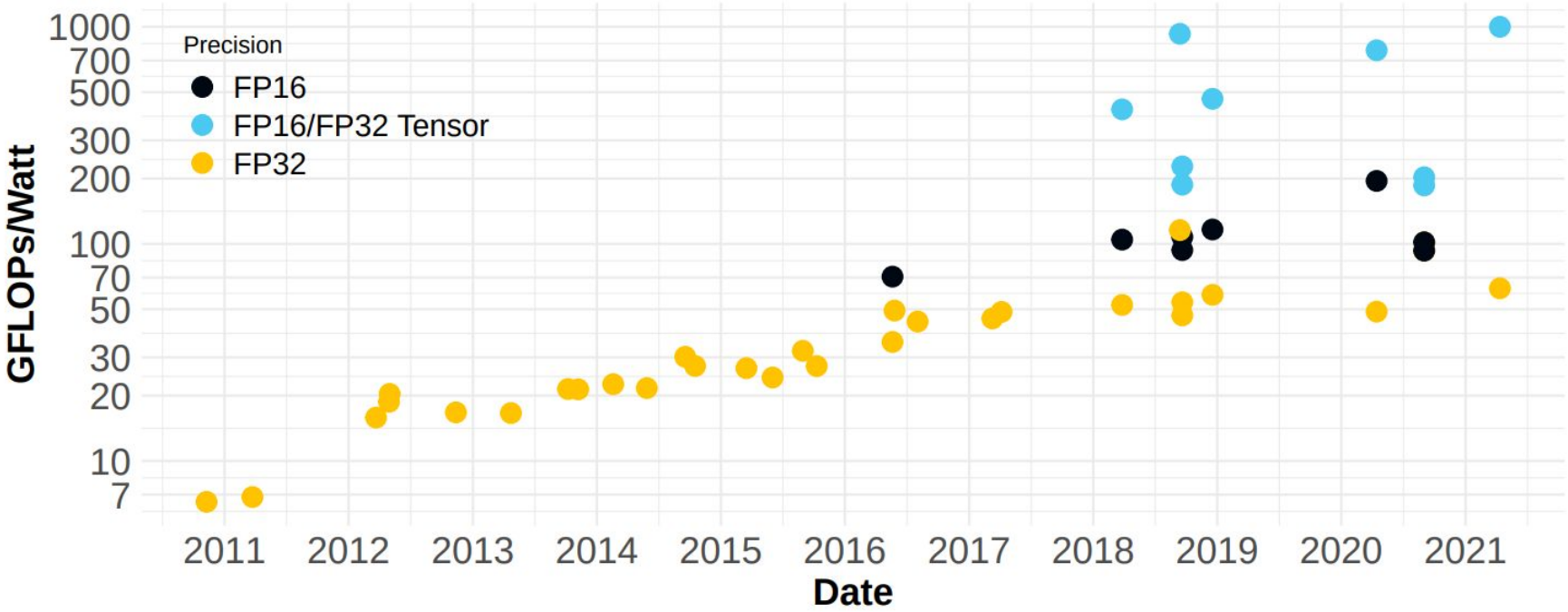
Сколько FLOP'ов уходит на умножение матриц?

Ответ: lmn умножений и l(m - 1)n сложений. Это 8 умножений для матриц 2×2.

Алгоритм Штрассена — 7 умножений

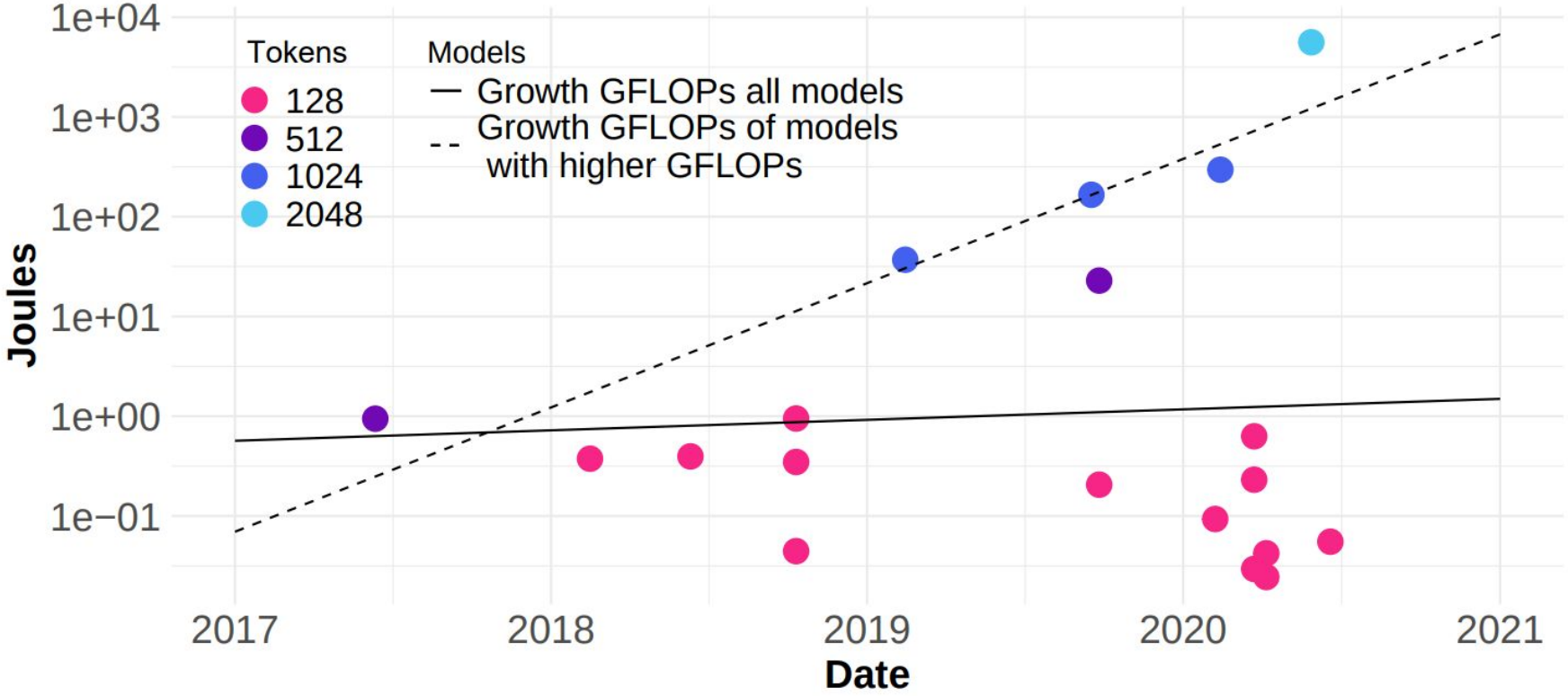
Ответ 2: на самом деле всего lmn FLOP'ов потому что fused multiply-add





Как изменялось теоретическое потребление мощности видеокарт Nvidia с течением времени

Источник: [arXiv:2109.05472]



Затраты энергии на один forward pass моделей для обработки естественного языка.

Источник: arXiv:2109.05472

Используемая память

```
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 30.00 MiB. GPU 0 has a total capacity
of 23.64 GiB of which 21.44 MiB is free. Process 581341 has 19.98 GiB memory in use. Process 1444241 has
3.63 GiB memory in use. Of the allocated memory 3.35 GiB is allocated by PyTorch, and 93.07 MiB is
reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting
max_split_size_mb to avoid fragmentation.  See documentation for Memory Management and
PYTORCH_CUDA_ALLOC_CONF
```

Узнали? Согласны?

- Для работы нейросети нужно загрузить в память: данные, веса модели.
- Для обучения модели дополнительно нужно сохранять: промежуточные результаты вычислений, градиент по параметром, статистики градиента
- По умолчанию все это хранится в оперативной памяти (RAM или VRAM).
- Но есть варианты, например, gradient checkpointing.

Используемая память

```
$ nvidia-smi
Sat May  4 17:04:26 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.129.03             Driver Version: 535.129.03   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA TITAN RTX               Off | 00000000:03:00.0 Off |                  N/A |
| 41%   34C    P8              23W / 280W |  20467MiB / 24576MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA TITAN RTX               Off | 00000000:04:00.0 Off |                  N/A |
| 41%   25C    P8               3W / 280W |      3MiB / 24576MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   2  NVIDIA TITAN RTX               Off | 00000000:82:00.0 Off |                  N/A |
| 41%   25C    P8               4W / 280W |      3MiB / 24576MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   3  NVIDIA TITAN RTX               Off | 00000000:83:00.0 Off |                  N/A |
| 41%   26C    P8              15W / 280W |      3MiB / 24576MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A    581341      C   python                                  20464MiB |
+---------------------------------------------------------------------------------------+
```

Скорость работы: latency VS throughput

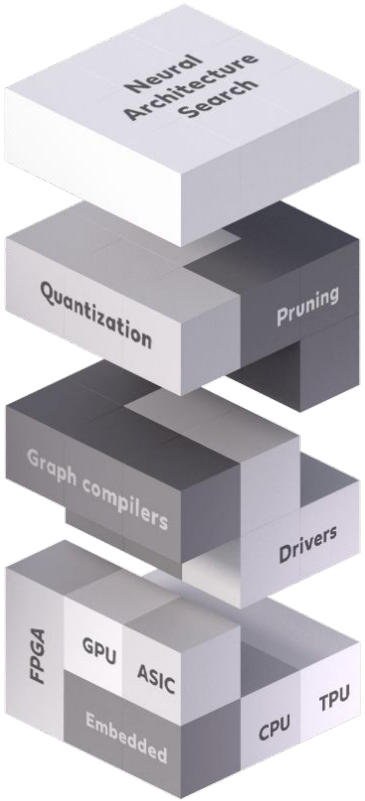
- Под скоростью работы часто имеют в виду разные вещи.
- Latency — время отклика, то есть время от воздействия до реакции.
- Throughput — пропускная способность, то есть сколько данных мы можем обработать за единицу времени.

Что выбрать для перевозки картошки? А пельменей?

latency optimized

throughput optimized

>>> Так как сделать DL эффективным?



LOW LEVEL: HARDWARE

- GPU
- TPU
- CPU
- FPGA
- Ascend

LOW LEVEL: CODE

- CUDA
- Graph Compilers
- MKL - DNN
- JIT
- TorchScript

HIGH LEVEL

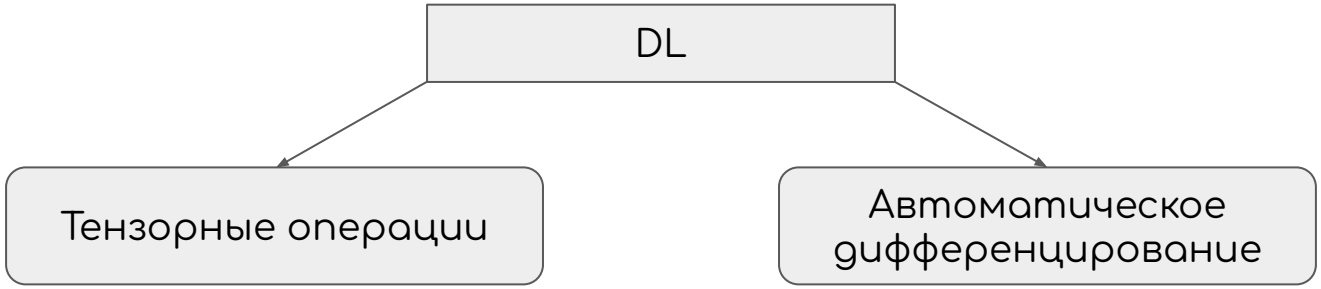
- Прунинг
- Квантизация
- Тензорные разложения
- Дистилляция

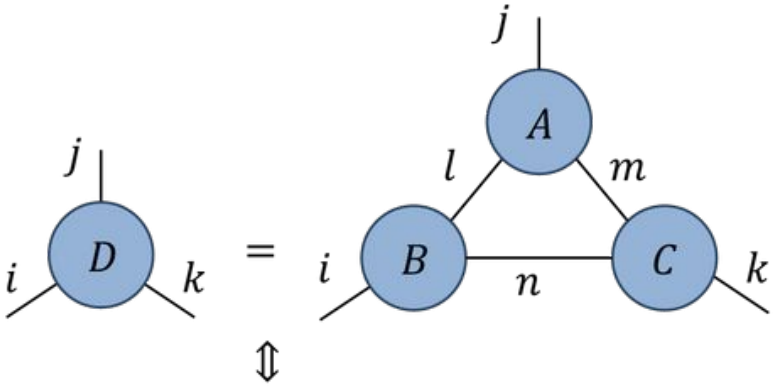
Эффективности можно достичь разными путями. В нашем курсе **основной упор будет на дизайн алгоритмов и архитектур, но про железо тоже немного поговорим.**

Image source: deci.ai

1. Введение.
2. Железо и инженерные подходы.
3. Квантизация.
4. Спарсификация, дистиляция.
5. Общие методы для более быстрого обучения и более эффективных моделей.
6. Тензорная декомпозиция.
7. Методы поиска архитектур (NAS).
8. Методы оптимизации LLM больших языковых моделей.

>>> Как работает DL

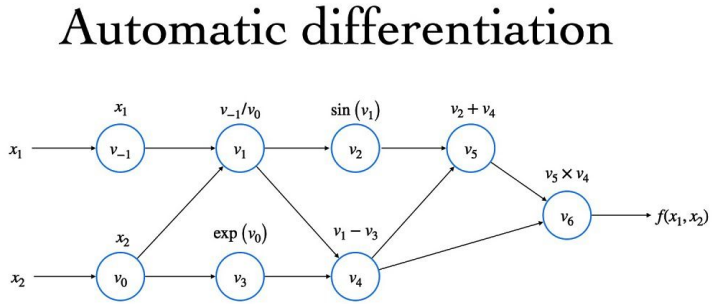




$$\Updownarrow$$

$$D_{ijk} = \sum_{lmn} A_{ljm} B_{iln} C_{nmk}$$

Источник: tensors.net/tutorial-1



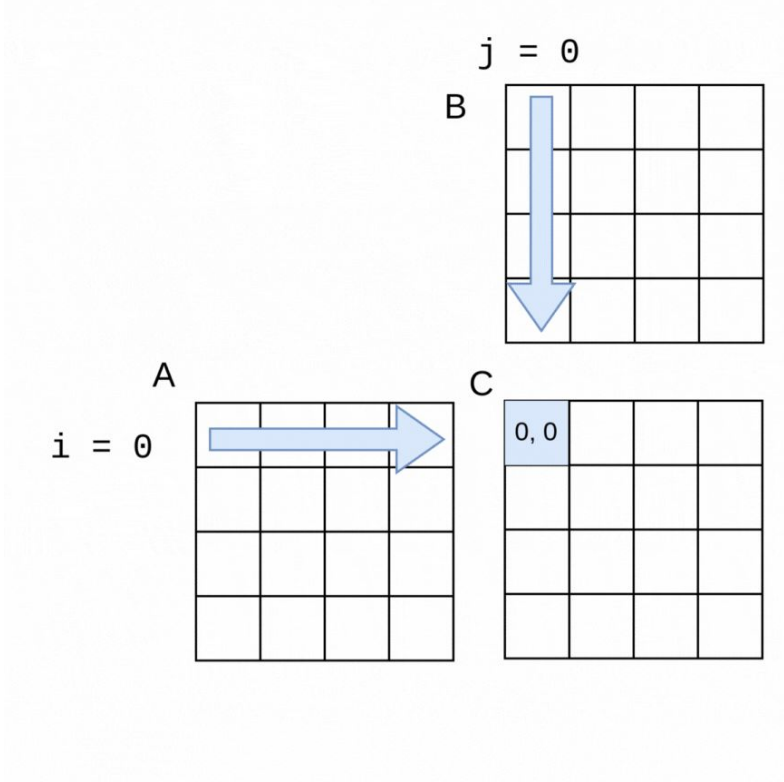
Источник: youtu.be/wG_nF1awSSY

Инженерные особенности:

- регулярный (но не последовательный) доступ к данным в памяти,
- операции сложения и умножения,
- возможность параллельных вычислений.

Математические особенности:

- хорошо изучены из-за простоты,
- непрерывность.



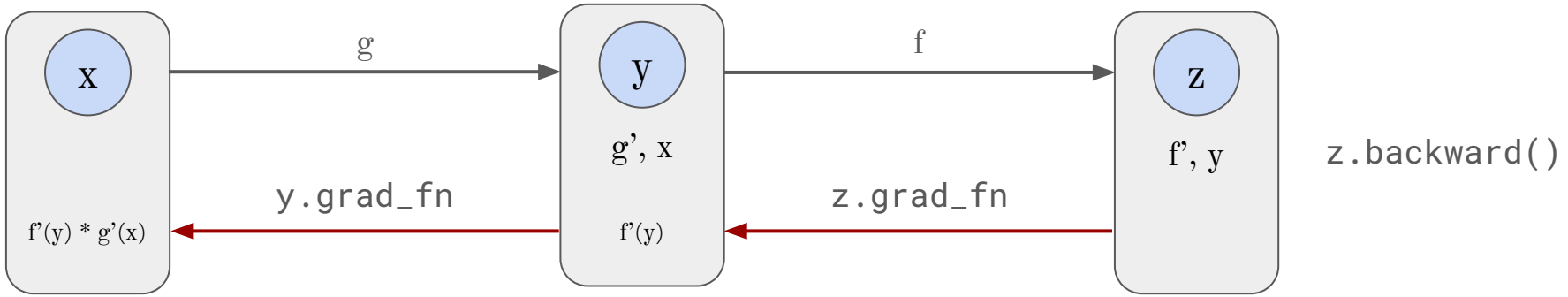
```
loss.backward()
optimizer.step()
```

Математически обучение нейронной сети — задача оптимизации, которая решается методом (улучшенного) градиентного спуска.

$$W_{t+1} = W_t - \alpha_{t+1} \left. \frac{\partial \text{Loss}(W, \text{input})}{\partial W} \right|_{W=W_t}$$

Как вычислить градиент автоматически? 2 алгоритма: forward mode и **reverse mode**.

$$\frac{d}{dx} f(g(x)) = \frac{d}{dx} g(x) \cdot \frac{d}{dy} f(y) \Big|_{y=g(x)}$$



- Выполняется сначала прямой проход по графу вычислений, затем обратный. Во время прямого прохода запоминается операция и результат вычислений.
- Затем граф обходится в обратном порядке и вычисляется производная сложной функции.

Как реализовать AD?

Динамический граф

- Граф вычислений строится динамически.
- Высокая гибкость.
- Меньше возможностей для оптимизации вычислений.
- Объектно-ориентированная парадигма.
- Библиотеки: PyTorch, MindSpore.
- PyTorch: тензор «запоминает», куда прокидывать градиент дальше.

Статический граф

- Граф вычислений компилируется.
- Меньшая гибкость.
- Больше возможностей для оптимизации вычислений.
- Функциональная парадигм.
- Библиотеки: JAX, MindSpore, TensorFlow.
- JAX: функция трассируется и строится статический граф.

>>> Спасибо за внимание!