Базовые методы ускорения

Феоктистов Дмитрий, @trandelik

На основе материалов
Максима Рябинина и
Егора Швецова

Материалы занятия

- Лекция Максима Рябинина

  https://github.com/mryab/efficient-dl-systems/blob/main/week03_fast_pipelines/lecture.pdf

- Лекция Егора Швецов

  https://github.com/On-Point-RND/Efficient-AI-Models/blob/main/Week%203/Lectures/Methods%20JIT%20and%20Compile.pdf

Как понять, что надо ускорять?

Как понять, что надо ускорять?

- Работает медленно

Как понять, что надо ускорять?

- Работает медленно
- GPU плохо утилизирована

Как понять, что GPU плохо утилизирована?

Как понять, что GPU плохо утилизирована?

```
sh-4.2$ nvidia-smi
Wed Dec 11 09:52:10 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000000:00:1E.0 Off |                    0 |
| N/A   44C    P0    60W / 149W |  11292MiB / 11441MiB |     13%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      9389      C   ...naconda3/envs/tensorflow_p36/bin/python 11012MiB |
|    0     21268      C   ...naconda3/envs/tensorflow_p36/bin/python   267MiB |
+-----------------------------------------------------------------------------+
```

Как понять, что GPU плохо утилизирована?

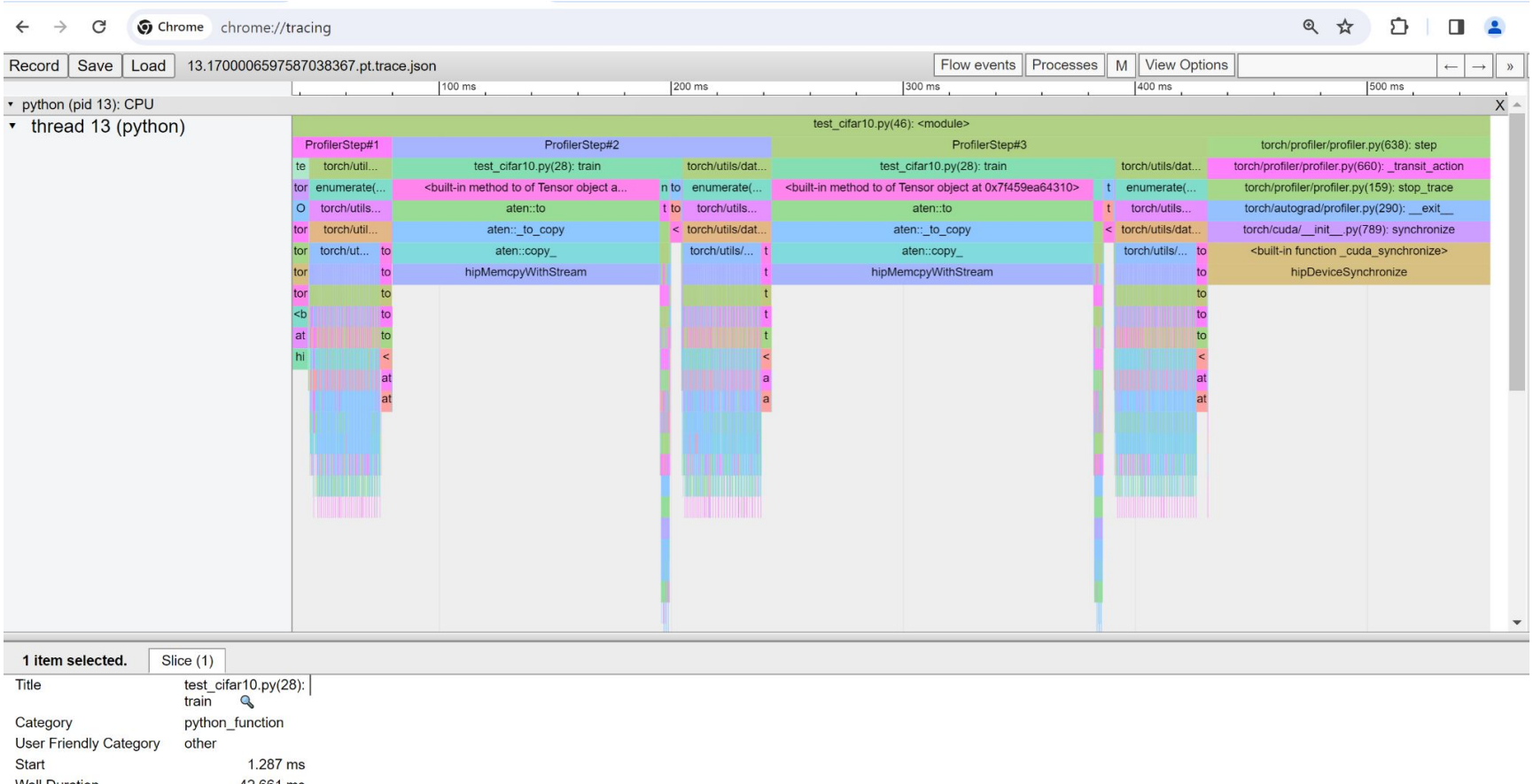
- SM утилизация
- $>$ 40 % $\Rightarrow$ пойдет
- nvidia-smi dmon

```
# gpu    pwr  gtemp  mtemp     sm    mem    enc    dec   mclk   pclk
# Idx      W      C      C      %      %      %      %    MHz    MHz
    0     60     34      -      0      0      0      0   7000   1350
    1     56     44      -      0      0      0      0   7000   1350
    0     60     34      -      1      0      0      0   7000   1350
    1     86     45      -      1      0      0      0   7000   1350
```

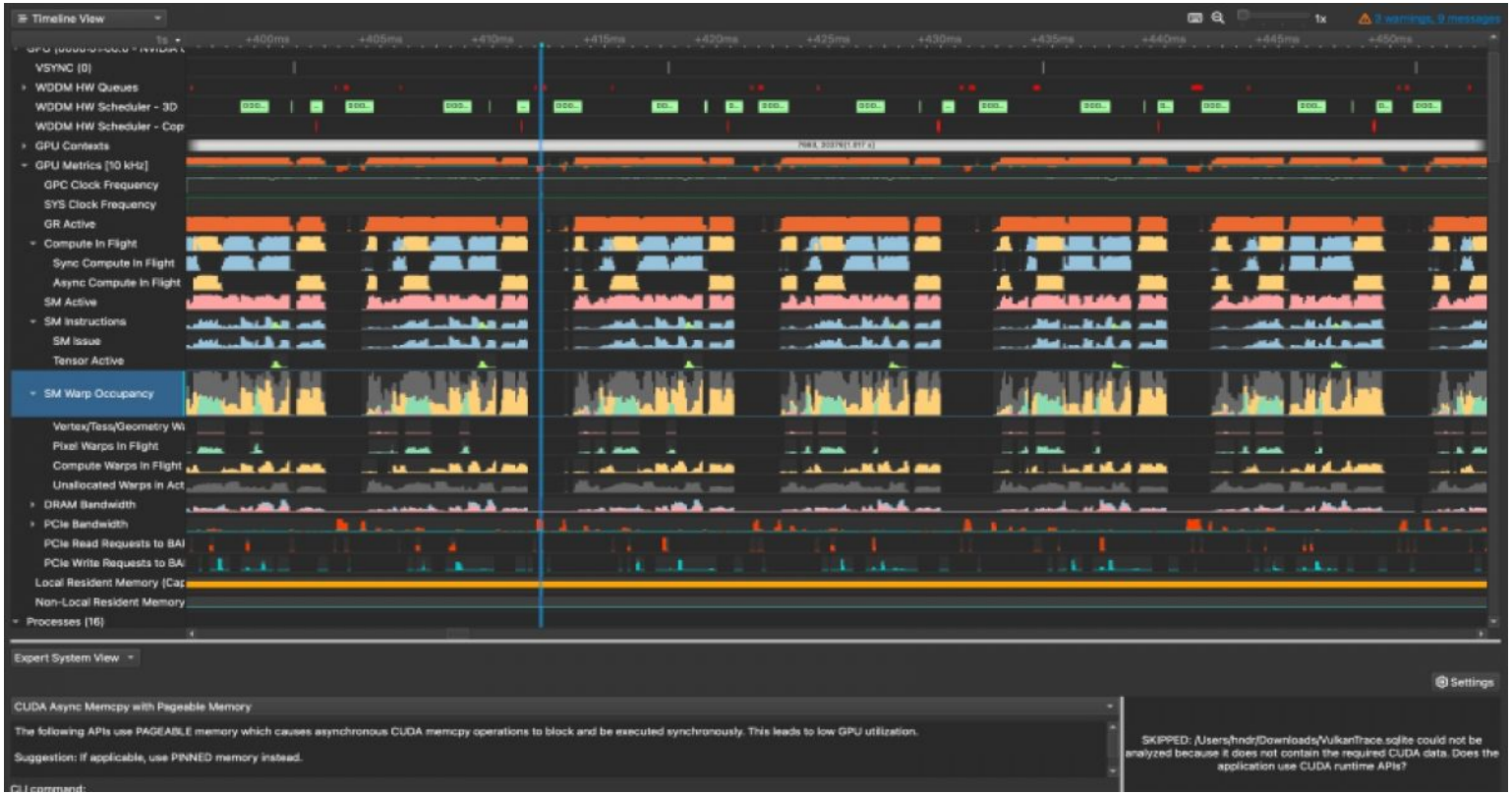
Как понять, что GPU плохо утилизирована?

- Model FLOPS utilization: отношение flops кода к теоретическому максимуму на данном железе
- MFU $>$ 45 % $\Rightarrow$ хорошо
- Надо считать руками, но есть неплохие аппроксимации

Что ускорять? (torch profiling)



Что ускорять? (nsight systems)



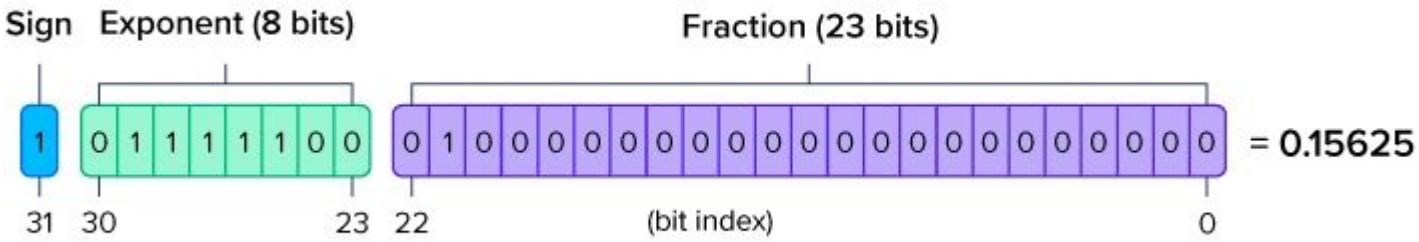
А как ускорять?

А как ускорять?

Давайте попробуем ускорить базовые операции - умножение матриц и свертки

Data types

Стандартный вариант хранения вещественных чисел - FP32



Data types

Может, стоит уменьшить точность?

Data types

Может, стоит уменьшить точность?

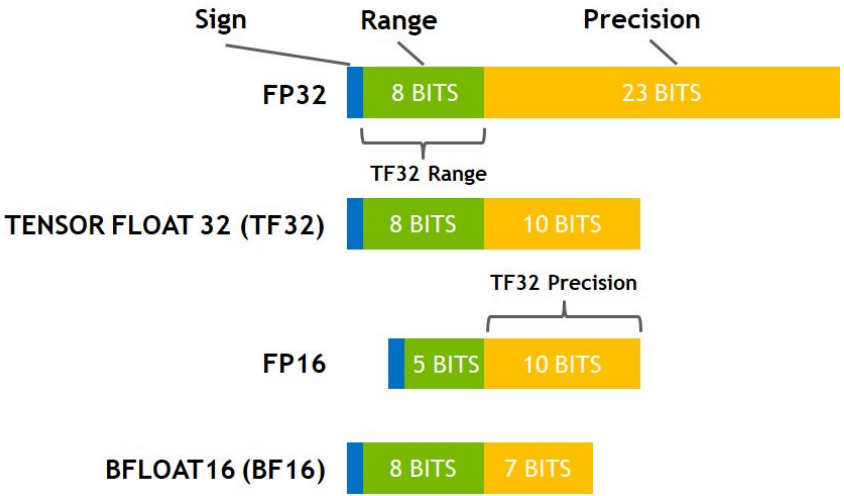
- Будет требоваться меньше памяти
- Быстрее: больше вычислений в те же ресурсы/нужно копировать меньше данных
- Можно использовать специальное железо

Data types

Может, стоит уменьшить точность?

- Будет требоваться меньше памяти
- Быстрее: больше вычислений в те же ресурсы/нужно копировать меньше данных
- Можно использовать специальное железо
- Незабываемый экспириенс с нестабильностью

Data types



- fp16 => CUDA 8+
- bf16 => современные GPU
- tf32 => Ampere+

https://developer.nvidia.com/blog/accelerating-ai-training-with-tf32-tensor-cores/

Data types

Tensor Cores:

- Специальный железный блок современных GPU (Volta+), который нацелен на перемножение матриц и свертки
- Специализируются на операции AB + C
- Живут в TF32, но при этом принимают и fp16/bf16 и даже int8/int4
- Все работает автоматически

Table 1. Tensor Core requirements by cuBLAS or cuDNN version for some common data precisions. These requirements apply to matrix dimensions M, N, and K.

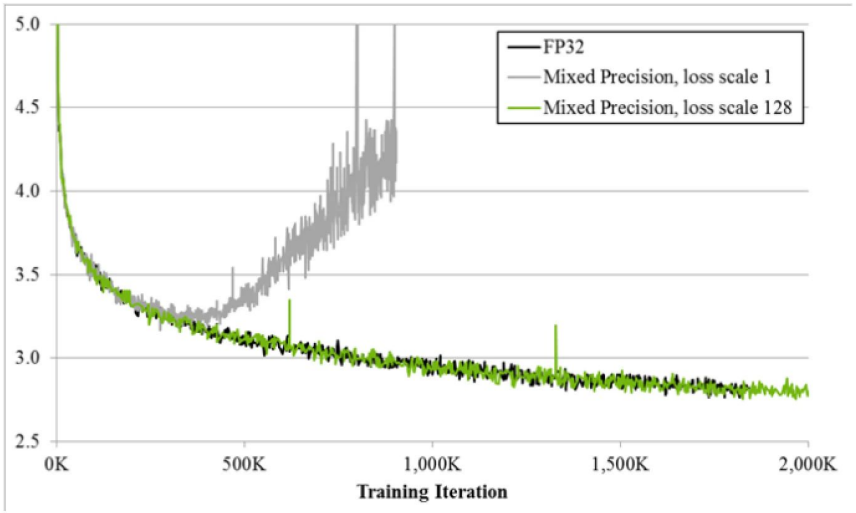
Tensor Cores can be used for...	cuBLAS version < 11.0 cuDNN version < 7.6.3	cuBLAS version ≥ 11.0 cuDNN version ≥ 7.6.3
INT8	Multiples of 16	Always but most efficient with multiples of 16; on A100, multiples of 128.
FP16	Multiples of 8	Always but most efficient with multiples of 8; on A100, multiples of 64.
TF32	N/A	Always but most efficient with multiples of 4; on A100, multiples of 32.
FP64	N/A	Always but most efficient with multiples of 2; on A100, multiples of 16.

https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html#requirements-tc

Data types

- Незабываемый экспириенс с нестабильностью

А как быть с этим?



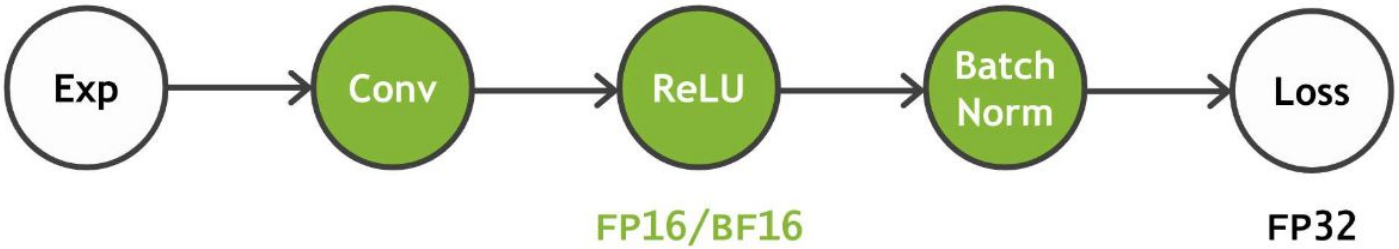
https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

Data types

- Просто в fp16 не выйдет
- Перемножать матрицы в fp16 еще можно, а вот softmax/layernom/batchnorm уже не стоит

Data types

- Просто в fp16 не выйдет
- Перемножать матрицы в fp16 еще можно, а вот softmax/layernom/batchnorm уже не стоит
  => mixed precision



Data types

А как быть с оптимизацией? Допустим, у Adam есть деление в шаге, что плохо для низкой точности $\Longrightarrow$ Будем хранить state в fp32. Итого:

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla\lfloor \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

Data types

А как быть с оптимизацией? Допустим, у Adam есть деление в шаге, что плохо для низкой точности => Будем хранить state в fp32. Итого:

- 2 байта на параметр
- 4 байт на мастер-параметр
- 8 байт на статистики
- 2/4 байта на градиент

Итого 16-18 байт на параметр => нет экономии на весе модели!

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla\lfloor \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

Data types

А как быть с оптимизацией? Допустим, у Adam есть деление в шаге, что плохо для низкой точности => Будем хранить state в fp32. Итого:

- 2 байта на параметр
- 4 байт на мастер-параметр
- 8 байт на статистики
- 2/4 байта на градиент

Итого 16-18 байт на параметр => нет экономии на весе модели!

Но есть на активациях и это важно!

Algorithm 1: The Basic Adam Optimizer

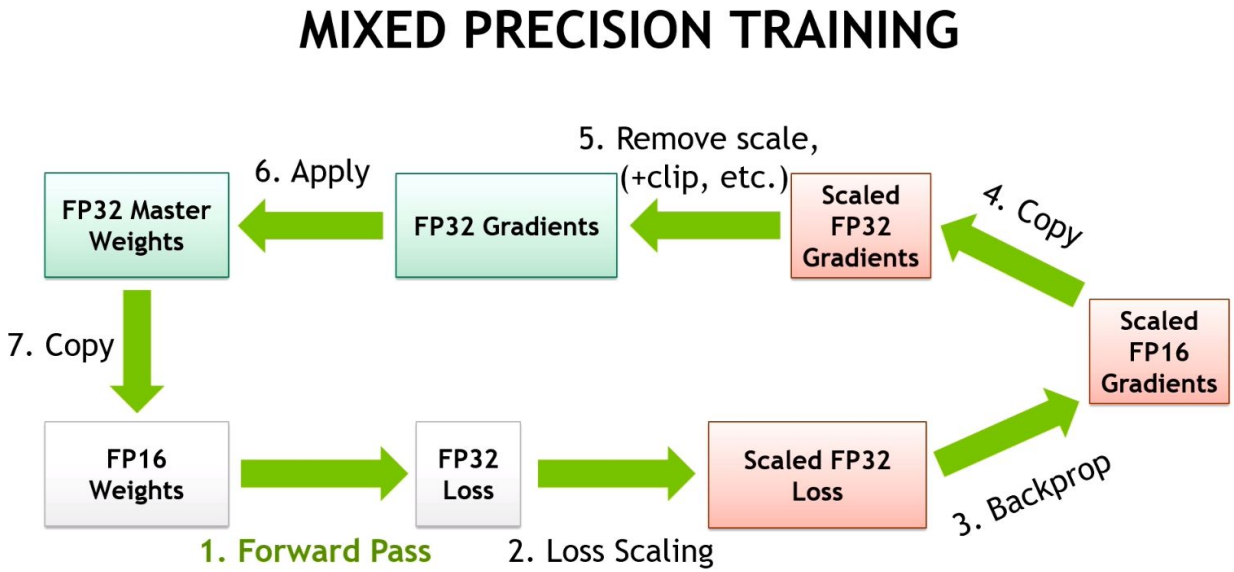
Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla \lfloor \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

Data types

А как быть с тем, что градиент в fp16 может быть 0/inf?

Data types

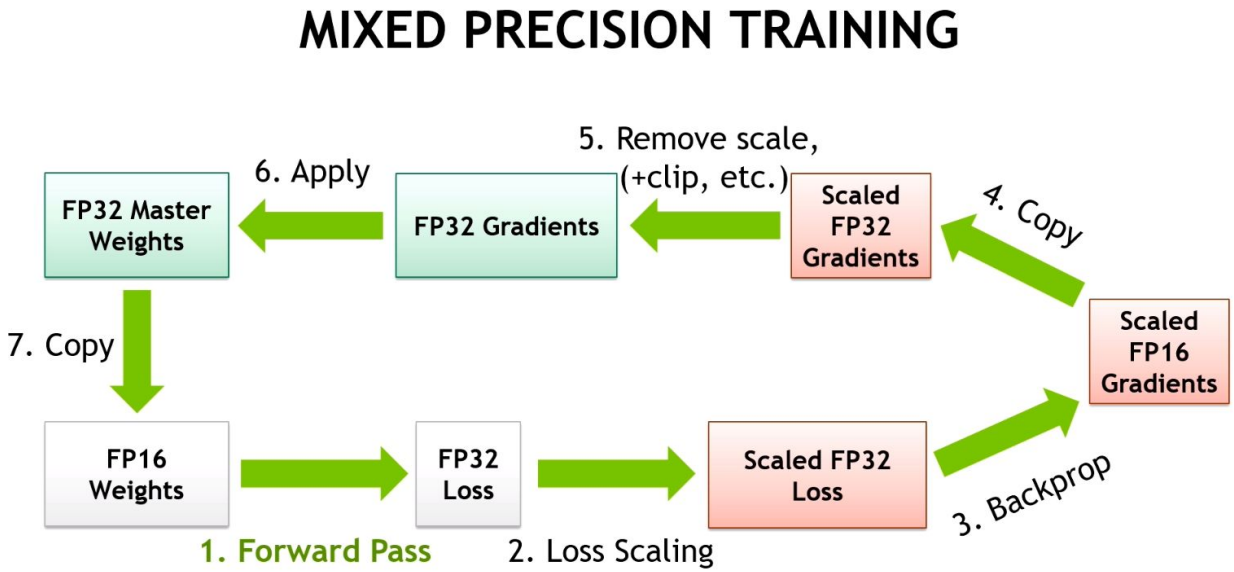
А как быть с тем, что градиент в fp16 может быть 0/inf?



https://developer.nvidia.com/blog/video-mixed-precision-techniques-tensor-cores-deep-learning/

Data types

А как быть с тем, что градиент в fp16 может быть 0/inf?

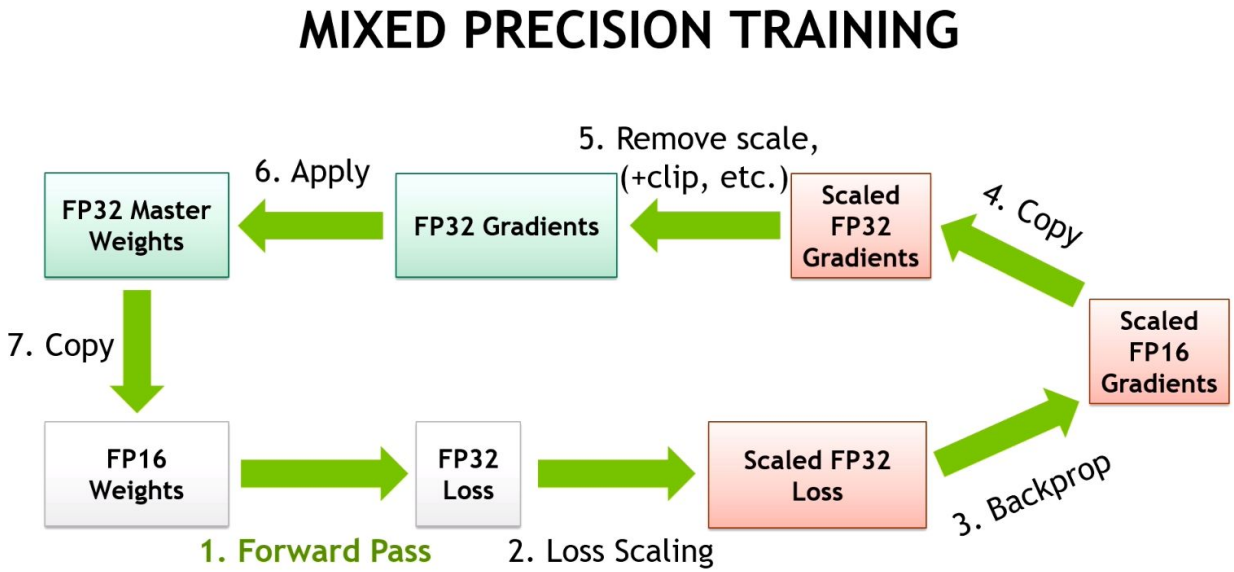


С bf16 scaling не нужен (почему?)

https://developer.nvidia.com/blog/video-mixed-precision-techniques-tensor-cores-deep-learning/

Data types

А как быть с тем, что градиент в fp16 может быть 0/inf?



С bf16 scaling не нужен (почему?), но более сложная схема нужна с fp8

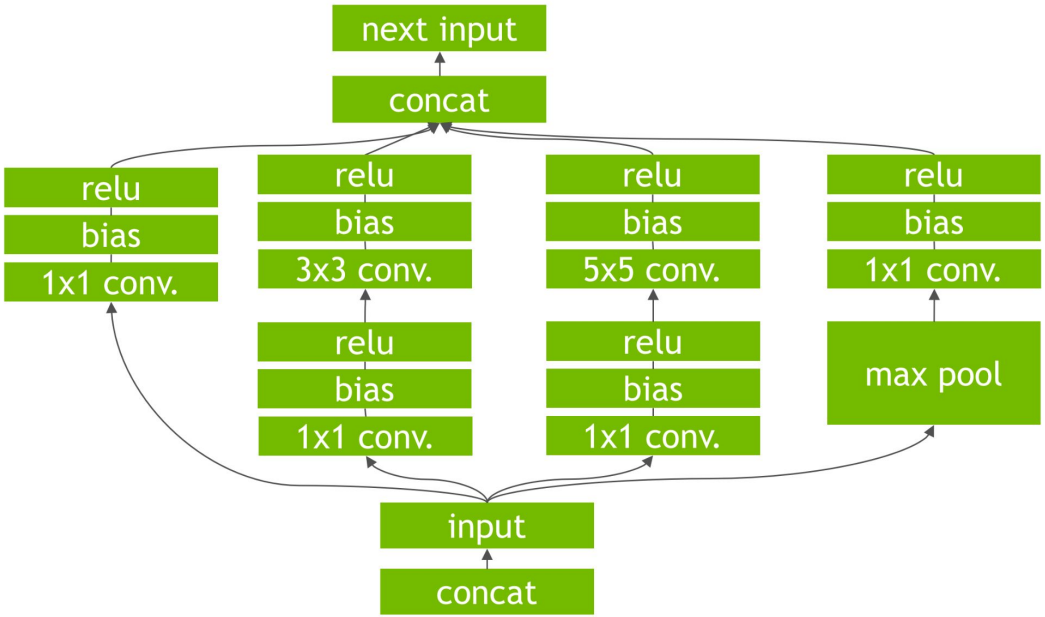
https://developer.nvidia.com/blog/video-mixed-precision-techniques-tensor-cores-deep-learning/

Может, ускорим что-то еще?

А как ускорять?

Давайте попробуем ускорить базовые операции - умножение матриц и свертки

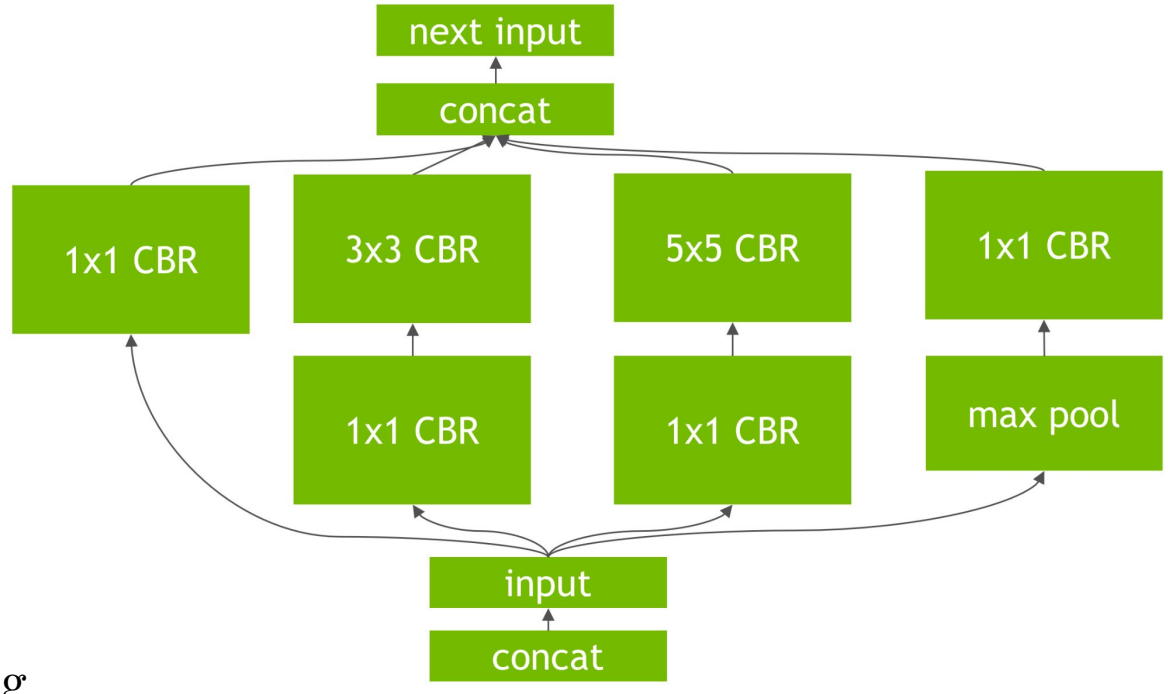
Kernel Fusing



Можно ли сэкономить на графе вычислений?

https://developer.nvidia.com/blog/production-deep-learning-nvidia-gpu-inference-engine/

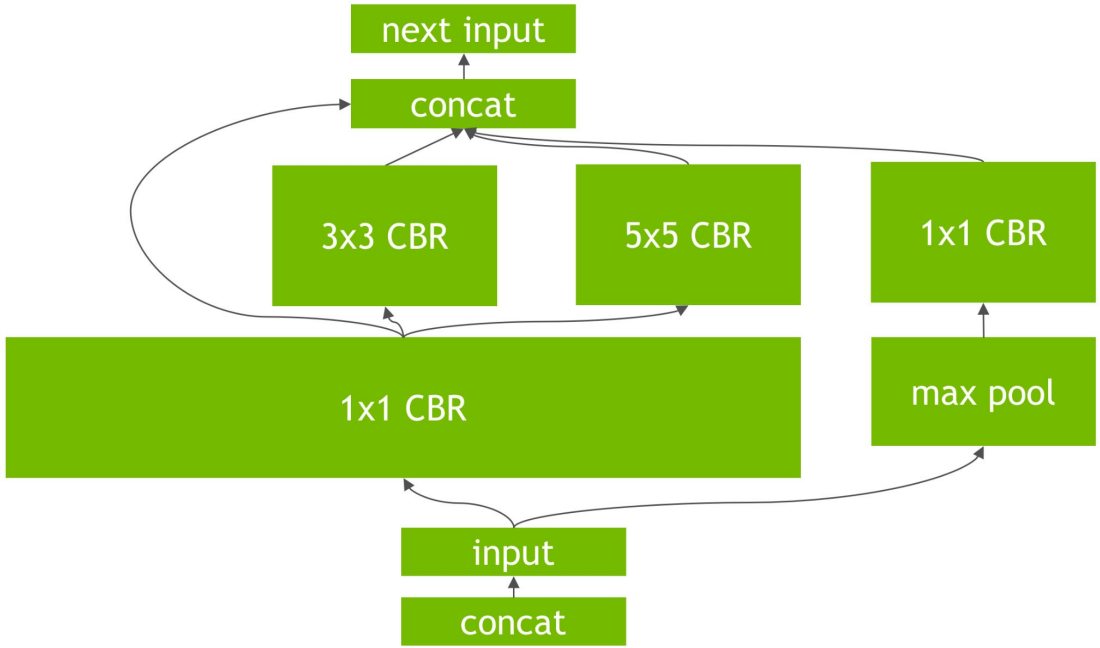
Kernel Fusing



Vertical Fusing

https://developer.nvidia.com/blog/production-deep-learning-nvidia-gpu-inference-engine/

Kernel Fusing



Horizontal Fusing

https://developer.nvidia.com/blog/production-deep-learning-nvidia-gpu-inference-engine/

Fusing example: conv + BN (at inference)

$$z = W * x + b$$

$$\mathrm{BN}(z) = \gamma \cdot \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\mathrm{BN}(W * x + b) = \gamma \cdot \frac{W * x + b - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\mathrm{BN}(W * x + b) = \alpha \cdot (W * x + b) + \beta - \alpha \cdot \mu = \alpha \cdot W * x + (\alpha \cdot b + \beta')$$

Activation fusing: cuBLAS предлагает специальный кернел

https://arikpoz.github.io/posts/2025-05-07-faster-models-with-graph-fusion-how-deep-learning-frameworks-optimize-your-computation/

Как такое провернуть?

- JIT компиляция -> преобразуем код в какое-то промежуточное представление и в процессе соптимизируем его

- torch.jit.trace: модель + образец данных = ускорение + замороженная логическая структура
- torch.jit.script: модель + статическая оптимизация содержимого nn.Module

```
def f(x, y):
    if x.sum() < 0:
        return -y
    return custom_cpp_library.f(y)
```

Как такое провернуть?

- PyTorch 2.0 = torch.compile

- **TorchDynamo** analyzes your PyTorch code and extracts a computation graph. This graph represents the sequence of operations in your code, identifying opportunities for optimization. It handles Python control flow and data-dependent operations by introducing "guards" that check for changes in input properties (like shape or data type) and trigger recompilation if necessary.

- **TorchInductor.** The extracted graph is then passed to TorchInductor, which further optimizes the graph and compiles it into highly efficient machine code, often leveraging specialized hardware features like Tensor Cores on GPUs. TorchInductor can generate kernels using various backends, including Triton, a high-performance language for writing GPU kernels.

Вывод

Вывод

- blackbox штука $=>$ мы не знаем, что там происходит внутри
- Универсальная штука $=>$ тупая, можно сильно лучше, если подумать

Ну давайте подумаем?

Ну давайте подумаем?

Может, ускорим популярные операции?

Optimizing Optimizers

- Наивное решение:

```
for param in params:
    # Retrieve necessary data for the current parameter
    # Perform operations (add, multiply, lerp, etc.)
    # Update the parameter
```

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla\lfloor \{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

https://christianjmills.com/posts/cuda-mode-notes/lecture-006/

Optimizing Optimizers

- Наивное решение:

```
for param in params:
    # Retrieve necessary data for the current parameter
    # Perform operations (add, multiply, lerp, etc.)
    # Update the parameter
```

- Foreach - параллельные операции над листом тензоров, но все еще нужно несколько кернелов (стандарт в PyTorch)

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla[\{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

https://christianjmills.com/posts/cuda-mode-notes/lecture-006/

Optimizing Optimizers

- Наивное решение:

```
for param in params:
    # Retrieve necessary data for the current parameter
    # Perform operations (add, multiply, lerp, etc.)
    # Update the parameter
```

- Foreach - зафьюеженые операции над листом тензоров, но все еще нужно несколько кернелов (стандарт в PyTorch)
- FusedAdam - один кернел (изначально было в Apex, но дошло и до PyTorch)

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla[\{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1). g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2). g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

https://christianjmills.com/posts/cuda-mode-notes/lecture-006/

Optimizing Optimizers

- Наивное решение:

```
for param in params:
    # Retrieve necessary data for the current parameter
    # Perform operations (add, multiply, lerp, etc.)
    # Update the parameter
```

- Foreach - зафьюеженые операции над листом тензоров, но все еще нужно несколько кернелов (стандарт в PyTorch)
- FusedAdam - один кернел (изначально было в Apex, но дошло и до PyTorch)
- В некоторых задачах дает ускорение в 30 процентов

Algorithm 1: The Basic Adam Optimizer

Require:
1: Initialized parameter θ_0, step size η, batch size N_B
2: Exponential decay rates $\beta_1, \beta_2, \varepsilon$ dataset $\{(x_i, y_i)\}_{i=1}^{N}$
Initialize: $m_0 = 0$, $v_0 = 0$
3: **For** all $t = 1, \ldots \ldots, T$ ***do***
4: Draw random batch $\{(x_{ik}, y_{ik})\}_{k=1}^{NB}$ from dataset
5: $g_t \leftarrow \sum_{k=1}^{N} \nabla[\{(x_{ik}, y_{ik}, \theta_{t-1})\}$ // $f'(\theta_{t-1})$
6: $m_t \leftarrow \beta_1 . m_{t-1} + (1-\beta_1).\ g_t$ // moving Average
7: $v_t \leftarrow \beta_2 . m_{t-1} + (1-\beta_2).g_t^2$
8: $\hat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}$, $\hat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$ // correction bias
9: $\theta_t \leftarrow \theta_{t-1} - \eta . \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon}$
10: **end for**
11: **return** final parameter θ_T

https://christianjmills.com/posts/cuda-mode-notes/lecture-006/

Optimizing Optimizers

- Доехало в torch.compile, но не до конца
- Работает для оптимизаторов, у которых есть foreach версия
- Не работает с, например, SparseAdam и L-BFGS
- Часто все равно нужно писать свои кернелы

```
optimizer = torch.optim.AdamW(model.parameters())

@torch.compile(fullgraph=False)
def compiled_step():
    optimizer.step()

# ... training loop
compiled_step()
```

Fused LayerNorm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Fused LayerNorm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- В PyTorch используется наивная реализация: два прохода для вычисления дисперсии, запуск нескольких кернелов

Fused LayerNorm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- В PyTorch используется наивная реализация: два прохода для вычисления дисперсии, запуск нескольких кернелов
- Nvidia APEX: Welford's алгоритм + fusing

$$\bar{x}_n = \frac{(n-1)\,\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$\sigma_n^2 = \frac{(n-1)\,\sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n} = \sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) - \sigma_{n-1}^2}{n}.$$

$$s_n^2 = \frac{n-2}{n-1}\,s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} = s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n} - \frac{s_{n-1}^2}{n-1}, \quad n > 1$$

https://nvidia.github.io/apex/layernorm.html

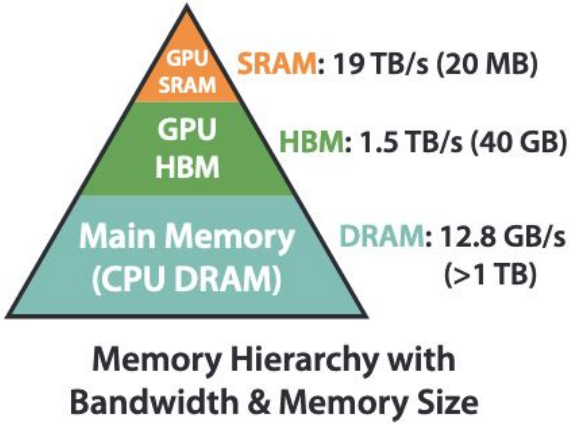
Flash Attention

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

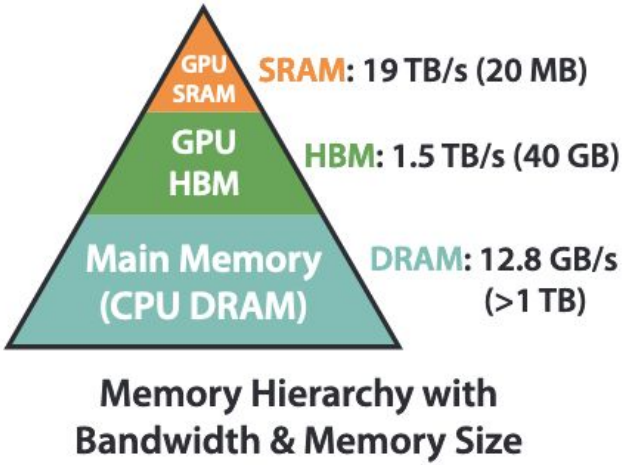
- O(Nd + N^2) обращений в HBM



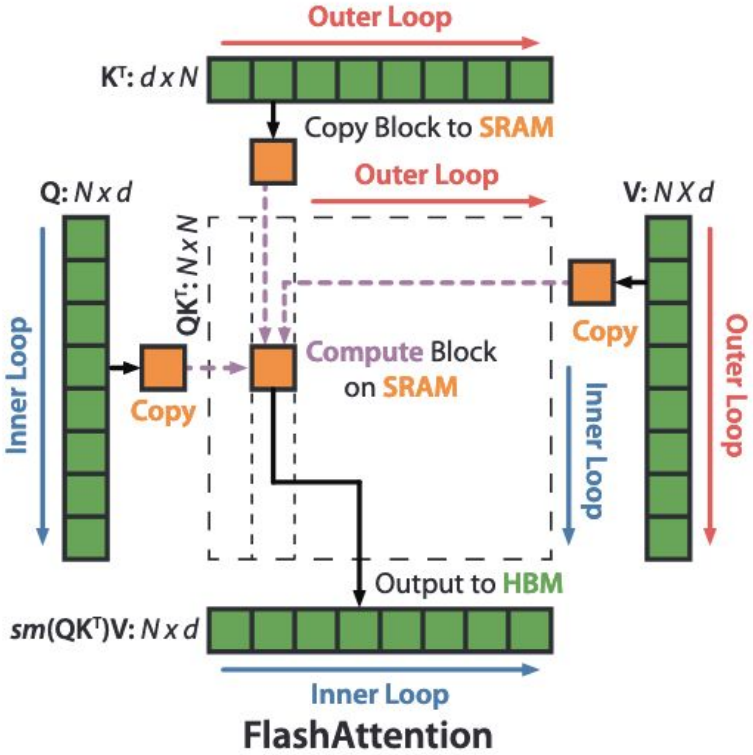
Memory Hierarchy with Bandwidth & Memory Size

https://arxiv.org/abs/2205.14135

Flash Attention



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention

https://arxiv.org/abs/2205.14135

Flash Attention

- Нарежем Q, K, V на блоки, влезающие в кэш (tiling)
- QK считаем через произведение блочных матриц
- Софтмакс можно тоже вычислять блочно

$$m(x) := \max_i \; x_i, \quad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix}, \quad \ell(x) := \sum_i f(x)_i, \quad \text{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

Как вычисляется softmax

For vectors $x^{(1)}, x^{(2)} \in \mathbb{R}^B$, we can decompose the softmax of the concatenated $x = \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \in \mathbb{R}^{2B}$ as:

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Слияние softmax из разных блоков

https://habr.com/ru/articles/669506/

Flash Attention

- На бекварде можно не хранить S, P (нужны для вычисления градиентов), если знать нормализационные факторы (см аппендикс статьи)
- Но нужно больше вычислений, так как мы будем пересчитывать некоторые вещи
- Зато меньше лезем в глобальную память

https://arxiv.org/abs/2205.14135

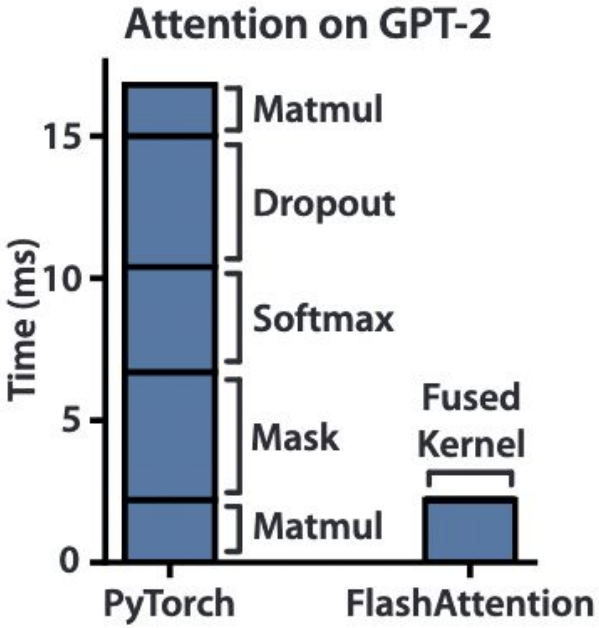
Flash Attention

- На бекварде можно не хранить S, P (нужны для вычисления градиентов), если знать нормализационные факторы (см аппендикс статьи)
- Но нужно больше вычислений, так как мы будем пересчитывать некоторые вещи
- Зато меньше лезем в глобальную память
- $O(Nd + N)$ обращений в HBM

https://arxiv.org/abs/2205.14135

Flash Attention

- Также пофьюзим кернелы



https://arxiv.org/abs/2205.14135

Flash Attention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

1: Set block sizes $B_r = \lceil \frac{M}{4d} \rceil$, $B_c = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.
2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$, $\ell = (0)_N \in \mathbb{R}^N$, $m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
3: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \ldots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \ldots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \ldots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
4: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \ldots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \ldots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \ldots, m_{T_r}$ of size B_r each.
5: **for** $1 \le j \le T_c$ **do**
6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
7: **for** $1 \le i \le T_r$ **do**
8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
10: On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
12: Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1}(\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
13: Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.
14: **end for**
15: **end for**
16: Return $\mathbf{O}$.

https://arxiv.org/abs/2205.14135

Flash Attention

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

https://arxiv.org/abs/2205.14135

Flash Attention

Model implementations	Context length	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Megatron-LM	1k	18.2	4.7 days (1.0×)
GPT-2 small - FlashAttention	1k	18.2	**2.7 days (1.7×)**
GPT-2 small - FlashAttention	2k	17.6	3.0 days (1.6×)
GPT-2 small - FlashAttention	4k	**17.5**	3.6 days (1.3×)

https://arxiv.org/abs/2205.14135

О чем мы забыли?

По секрету всему свету

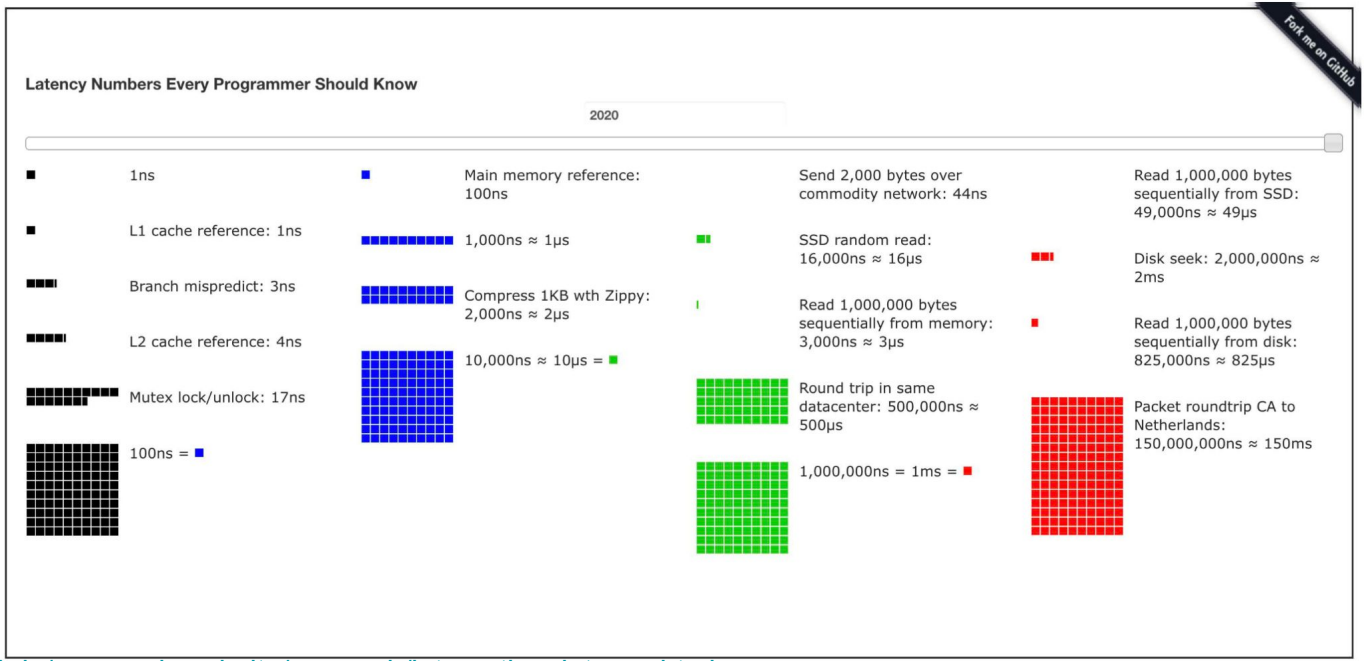
- DL - наука о данных

По секрету всему свету

- DL - наука о данных
- И даже для ускорения часто важнее всего они

По секрету всему свету

- DL - наука о данных
- И даже для ускорения часто важнее всего они



https://colin-scott.github.io/personal_website/research/interactive_latency.html

И как нам с этим жить?

Как хранить данные?

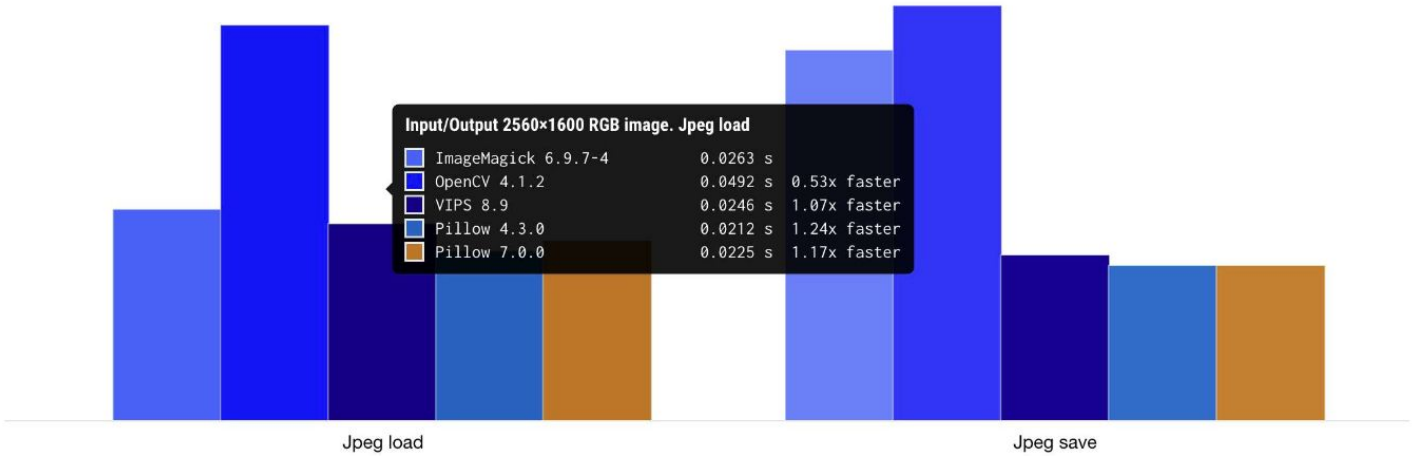
- Сырые данные (тексты/картинки) удобны для изучения, но неэффективны с точки зрения хранения
- Они часто занимают много места (много читаем с диска)
- Они требуют нетривиальную обработку

Как хранить данные?

- Сырые данные (тексты/картинки) удобны для изучения, но неэффективны с точки зрения хранения
- Они часто занимают много места (много читаем с диска)
- Они требуют нетривиальную обработку
- Ее можно сделать до начала обучения. Например:
  - Для картинок можно сделать заранее преобразования, которые не будут случайными в обучении
  - Для текстов можно сделать токенизацию заранее

Загрузка картинок

- Обычно это делается с помощью PIL.image
- Это медленно
- Можно сильно лучше, если использовать jpegturbo или nvJPEG

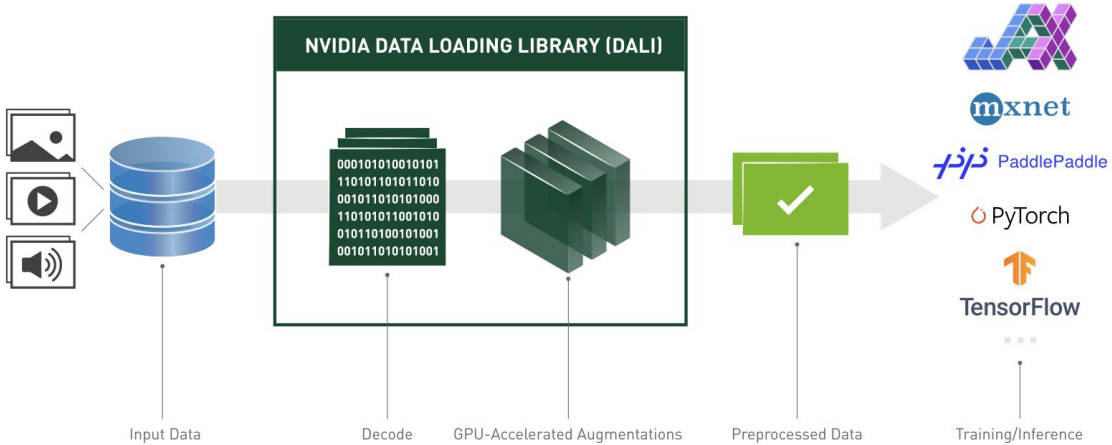


Предобработка картинок

- В PyTorch функции предобработки и аугментаций сделаны на CPU
- Это медленно!

Предобработка картинок

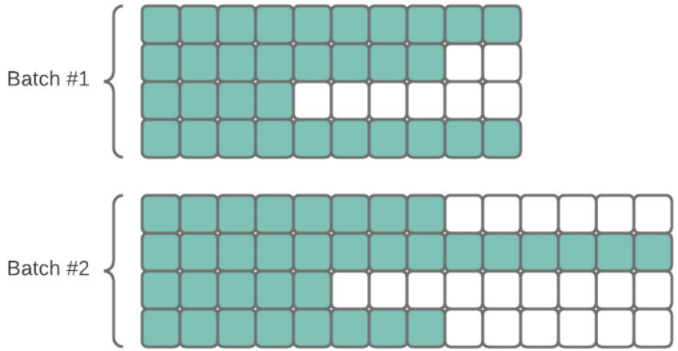
- В PyTorch функции предобработки и аугментаций сделаны на CPU
- Это медленно!
- Большинство из них можно перенести на GPU
- Есть библиотека NVIDIA DALI
- Подобное верно и для других доменов (аудио, видео, даже таблички и графы)



https://developer.nvidia.com/dali

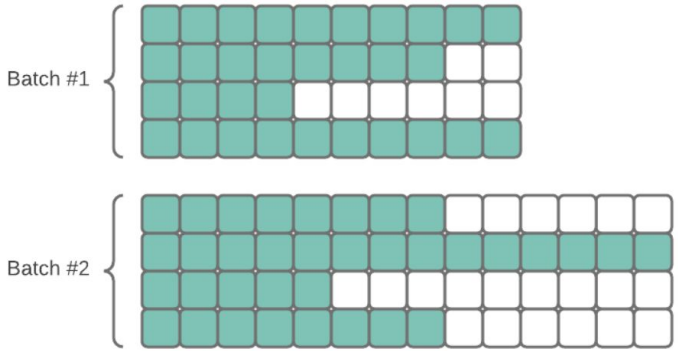
Предобработка текстов

- Нужно делать padding в батче, это занимает время. Можно ускорять



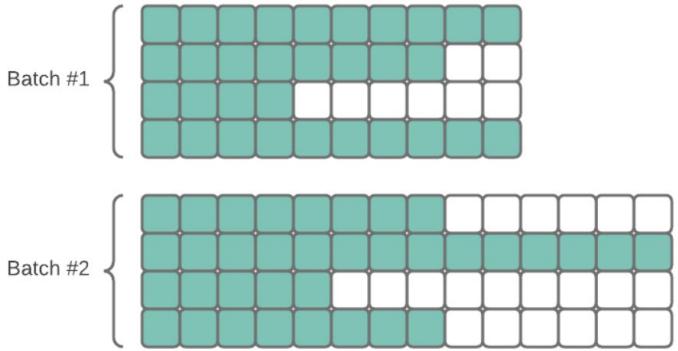
Предобработка текстов

- Нужно делать padding в батче, это занимает время. Можно ускорять
- Паддинг всего датасета => ужас



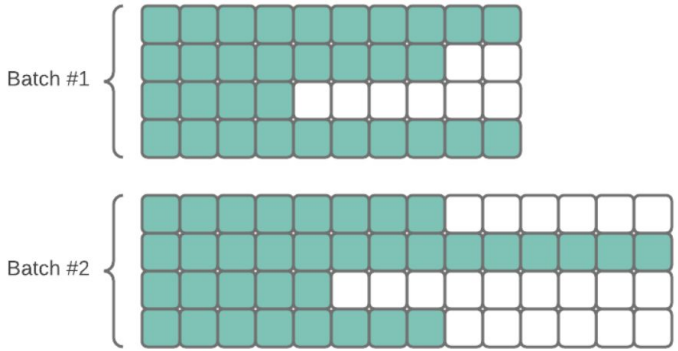
Предобработка текстов

- Нужно делать padding в батче, это занимает время. Можно ускорять
- Паддинг всего датасета => ужас
- Паддинг в collate_fn => лучше



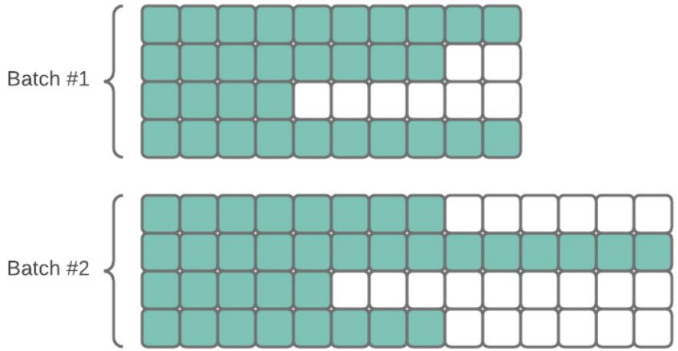
Предобработка текстов

- Нужно делать padding в батче, это занимает время. Можно ускорять
- Паддинг всего датасета => ужас
- Паддинг в collate_fn => лучше
- Можно объединять последовательности похожей длины в один батч
- Обучение теперь станет не iid, но зато можем сильно ускориться



Предобработка текстов

- Нужно делать padding в батче, это занимает время. Можно ускорять
- Паддинг всего датасета => ужас
- Паддинг в collate_fn => лучше
- Можно объединять последовательности похожей длины в один батч
- Обучение теперь станет не iid, но зато можем сильно ускориться
- А можно объединять короткие последовательности в одну
- Подобное верно и для других доменов с последовательностями



О чем важно не забывать при ускорении работы с данными?

О чем важно не забывать при ускорении работы с данными?

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

https://docs.pytorch.org/docs/stable/data.html

О чем важно не забывать при ускорении работы с данными?

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

- num_workers

https://docs.pytorch.org/docs/stable/data.html

О чем важно не забывать при ускорении работы с данными?

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

- num_workers
- pin_memory (физически выходы dataloader будут складываться в одном месте в RAM, что убирает приколы с виртуализацией и позволяет быстрее загружать данные на GPU и даже делать это асинхронно)

https://docs.pytorch.org/docs/stable/data.html

О чем важно не забывать при ускорении работы с данными?

```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

- num_workers
- pin_memory (физически выходы dataloader будут складываться в одном месте в RAM, что убирает приколы с виртуализацией и позволяет быстрее загружать данные на GPU и даже делать это асинхронно)
- prefetch_factor

https://docs.pytorch.org/docs/stable/data.html

Куда еще смотреть?

Куда еще смотреть?

CPU&GPU синхронизации!

Куда еще смотреть?

CPU&GPU синхронизации!

- CPU и GPU могут выполняться асинхронно, то есть пока вы делаете, например, backward можно подгружать следующий батч

Куда еще смотреть?

CPU&GPU синхронизации!

- CPU и GPU могут выполняться асинхронно, то есть пока вы делаете, например, backward можно подгружать следующий батч
- Синхронизации убивают это
- Например, loss.item() делает ее

Куда еще смотреть?

CPU&GPU синхронизации!

- CPU и GPU могут выполняться асинхронно, то есть пока вы делаете, например, backward можно подгружать следующий батч
- Синхронизации убивают это
- Например, loss.item() делает ее
- А loss.detach() нет

Куда еще смотреть?

CPU&GPU синхронизации!

- CPU и GPU могут выполняться асинхронно, то есть пока вы делаете, например, backward можно подгружать следующий батч
- Синхронизации убивают это
- Например, loss.item() делает ее
- А loss.detach() нет
- Вычисление метрик на sklearn
- Синхронизация + перекладывание в numpy + метрика на CPU

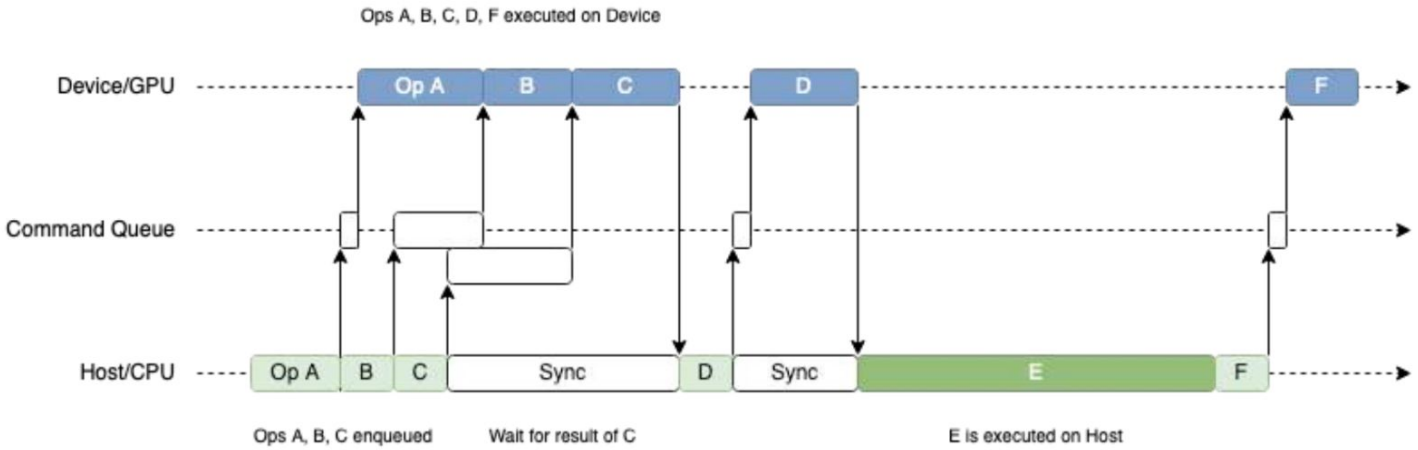
Куда еще смотреть?

CPU&GPU синхронизации!

- CPU и GPU могут выполняться асинхронно, то есть пока вы делаете, например, backward можно подгружать следующий батч
- Синхронизации убивают это
- Например, loss.item() делает ее
- А loss.detach() нет
- Вычисление метрик на sklearn
- Синхронизация + перекладывание в numpy + метрика на CPU

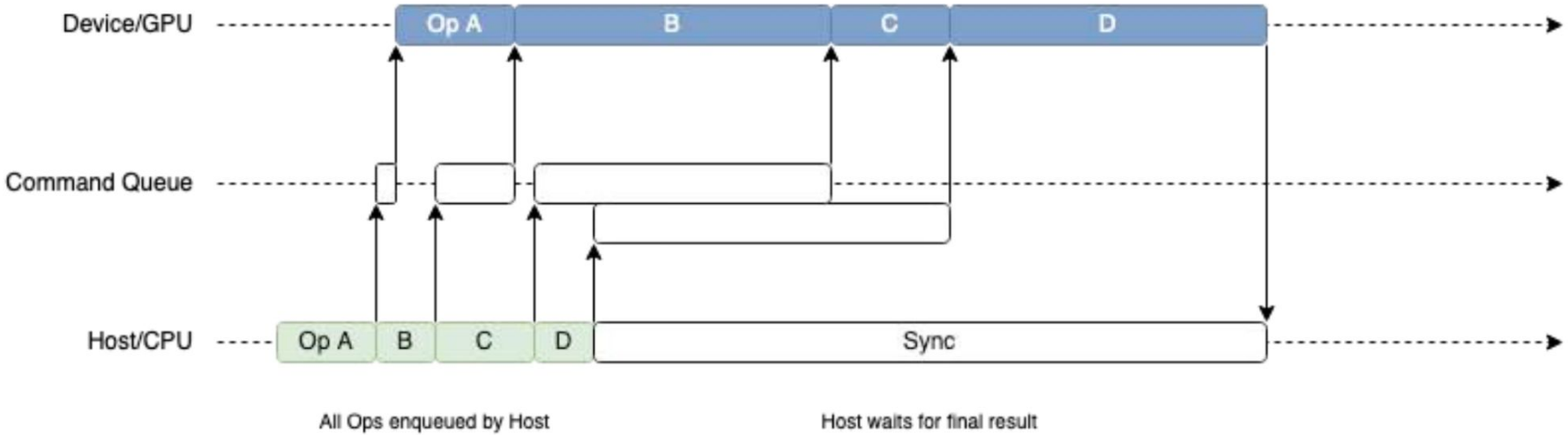
batch_size (я серьезно)

Квиз “Как это ускорить?”



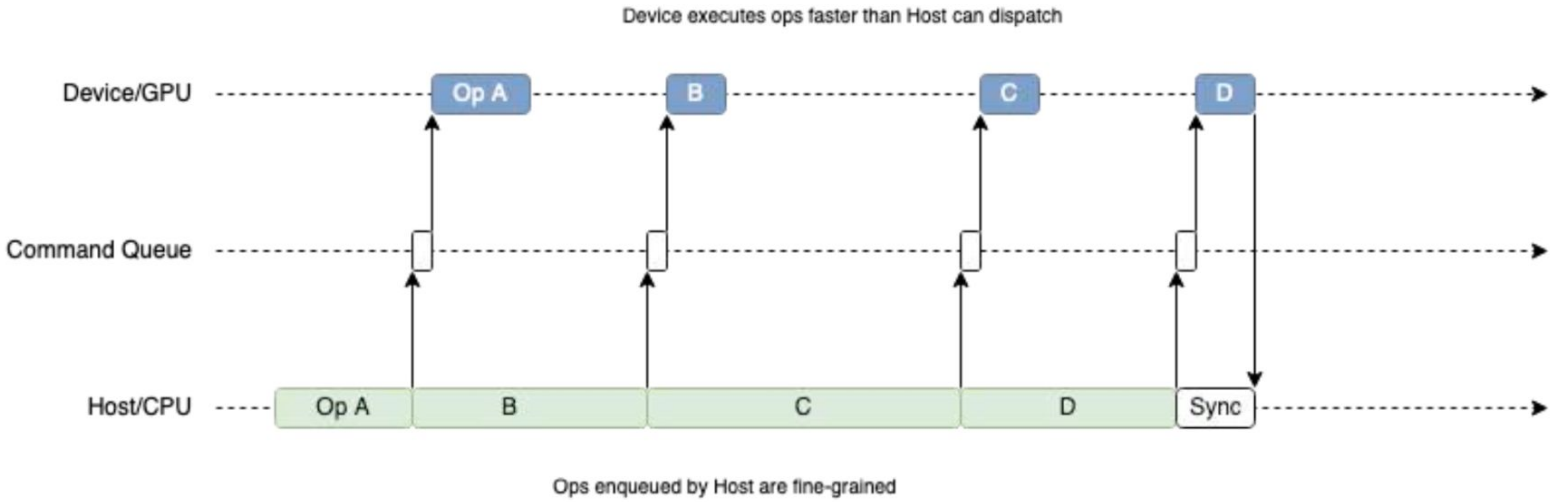
https://paulbridger.com/posts/nsight-systems-systematic-optimization/

Квиз “Как это ускорить?”



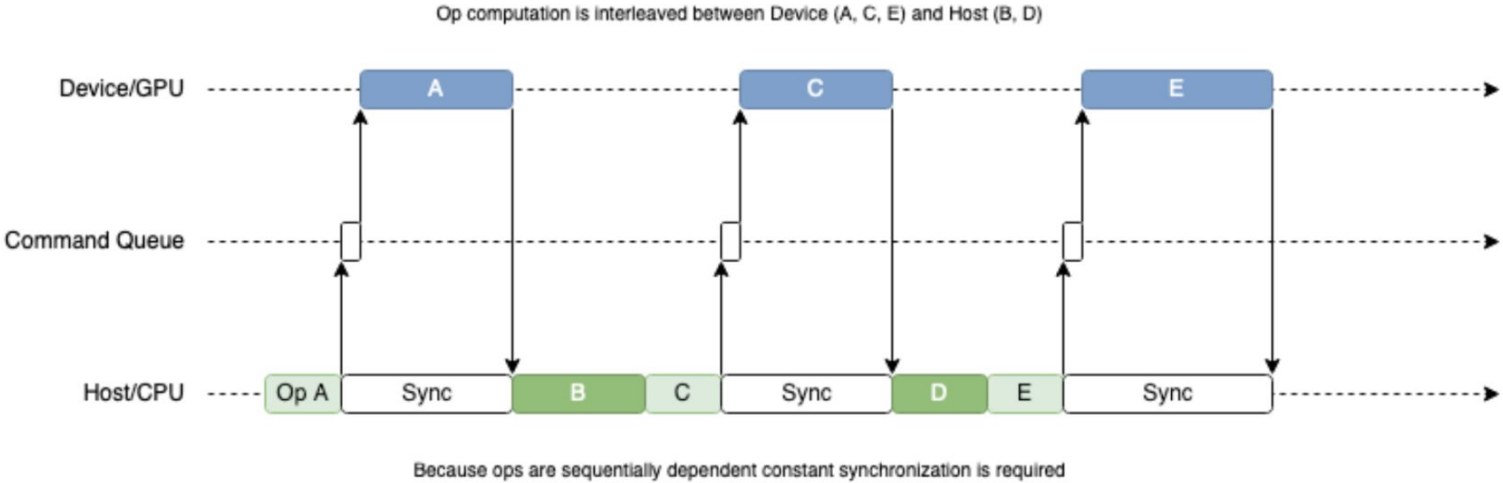
https://paulbridger.com/posts/nsight-systems-systematic-optimization/

Квиз “Как это ускорить?”



https://paulbridger.com/posts/nsight-systems-systematic-optimization/

Квиз “Как это ускорить?”



https://paulbridger.com/posts/nsight-systems-systematic-optimization/