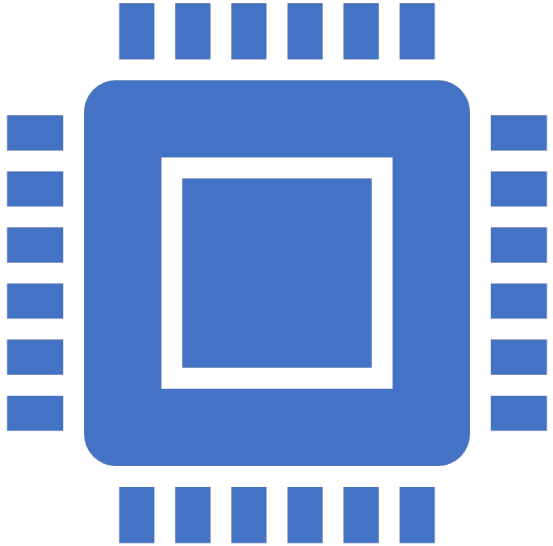
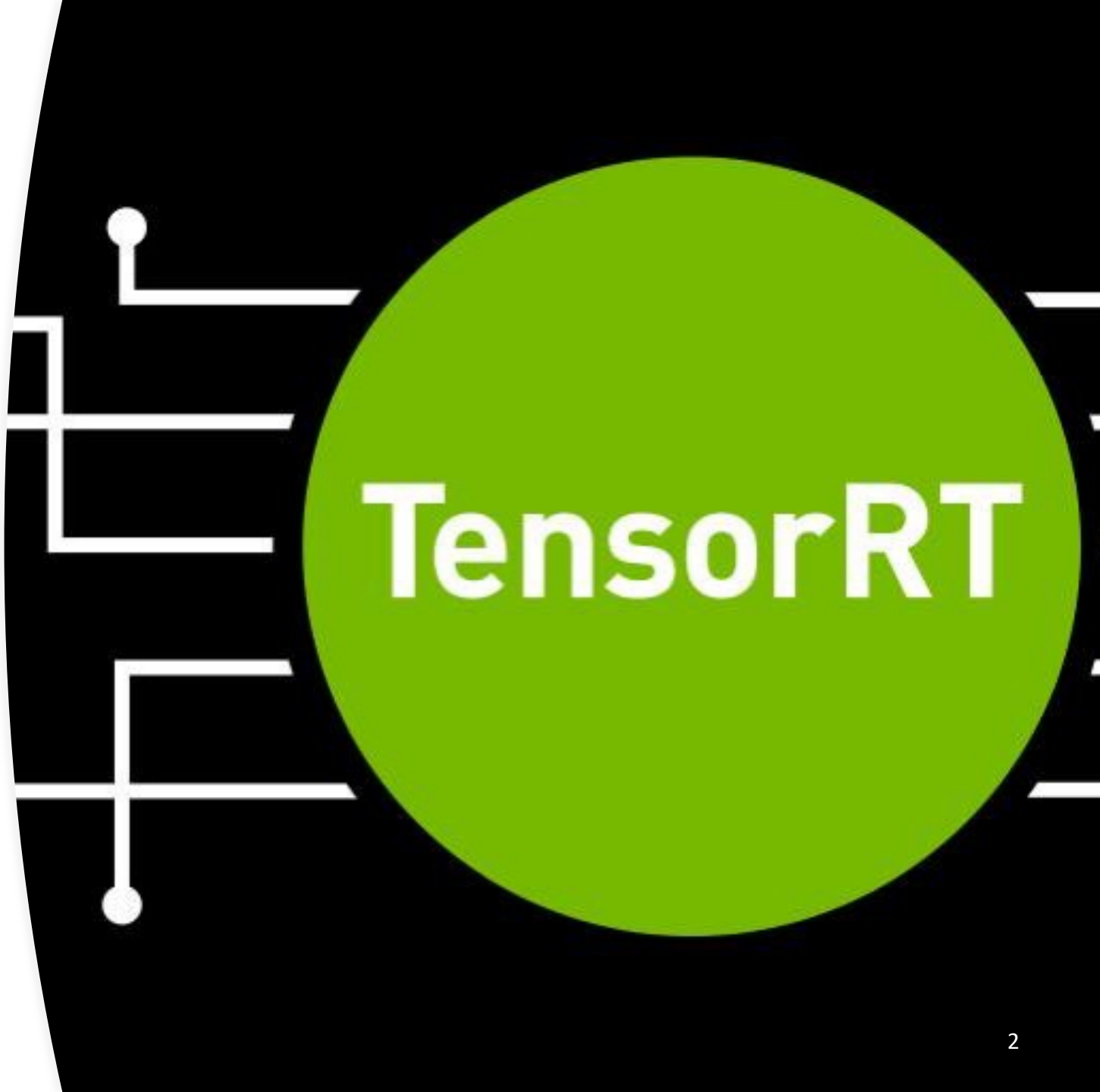


Оптимизация нейронных сетей

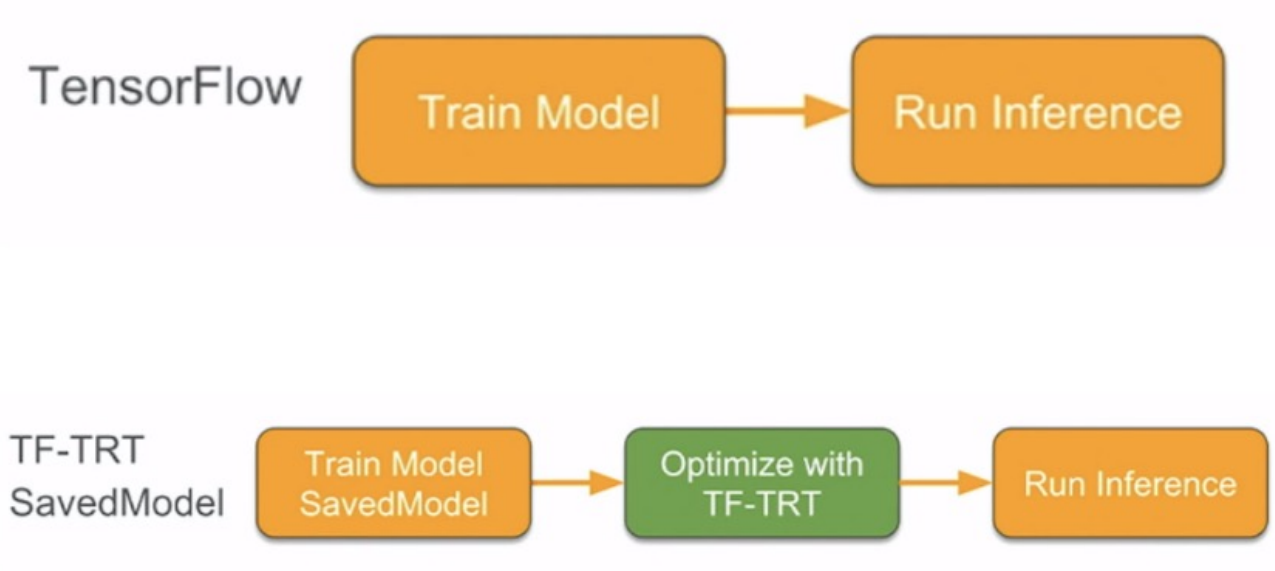
О чем поговорим

- Типы оптимизаций в TensorRT
- Использованием TF-TRT для оптимизации моделей tensorflow
- ONNX
- Как оптимизировать Pytorch



TensorRT

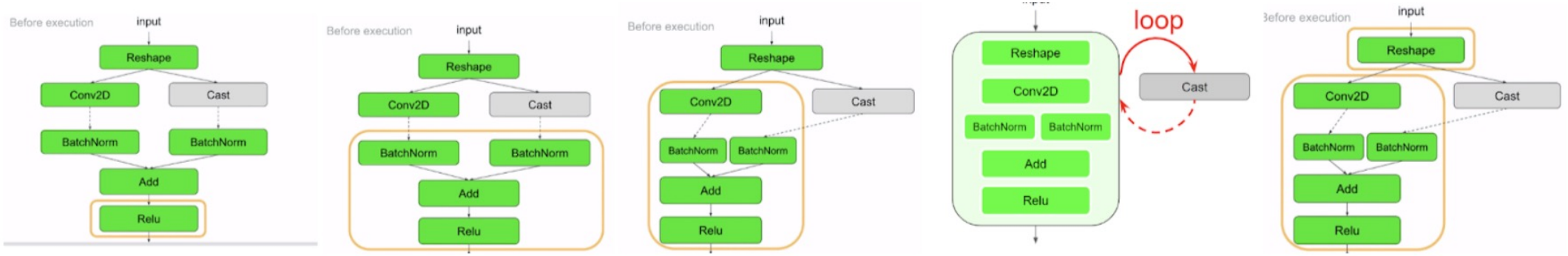
Что такое TensorRT? NVIDIA TensorRT это высокопроизводительный оптимизатор инференса моделей и среда выполнения, которые могут быть использованы для инференса с низкой точностью чисел (FP16, INT8) на GPU.
TensorRT сейчас интегрирован в Tensorflow и его использование заметно упрощается.
Помимо отпитизации также удаляется все лишнее в модели, что не пригодится для инференса.



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

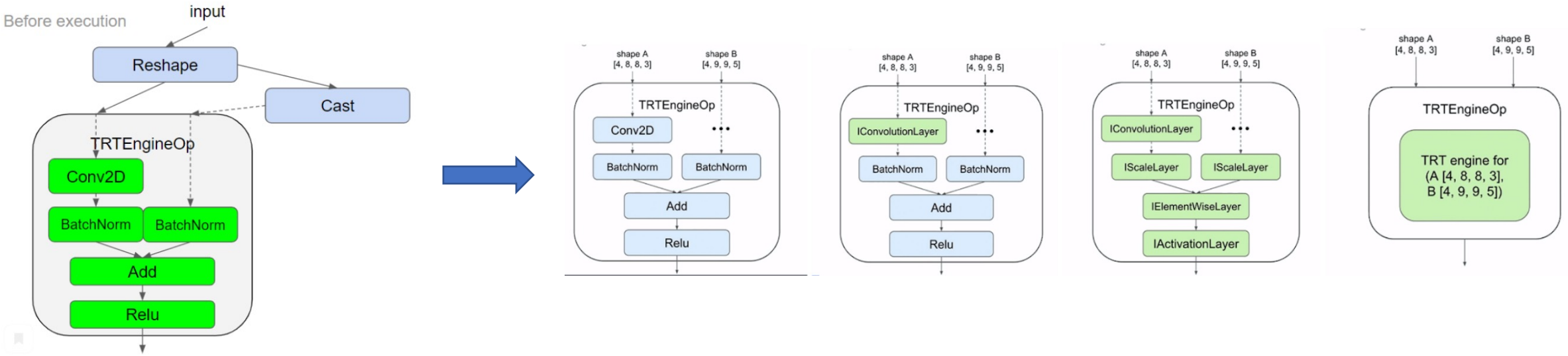
TensorRT оптимизация графа

1) TensoRT сканирует граф сети и определяет те места, которые он может оптимизировать
2) Преобразует те участки графа, которые ему известны (что не знает, оставляет на движке tensorflow)
3) Оптимизированные подграфы заменят собой сходные части графа модели



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

TensorRT оптимизация графа



https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

TensorRT оптимизация графа

Оптимизация в TensorRT – это баланс между скоростью и расходами на создание подграфа с использованием движка TensorRT. Что нужно контролировать?

minimum_segment_size – минимальный размер подграфа для оптимизации. Рекомендуется брать стандартно значение равное 3. Меньше\больше не всегда приводит к улучшению результата, но можно попробовать.
is_dynamic_op – если все размеры входных параметров известны, то ставим False и оптимизация будет идти лучше.
max_batch_size – максимальный размер батча, под который также будет оптимизирована архитектура. Не рекомендуется брать большим.
max_workspace_size_bytes – операторам TF-TRT часто требуется временное рабочее пространство. Параметр ограничивает это пространство и если места будет слишком мало, то может быть не найдено хорошей оптимизации.
precision_mode – параметр устанавливает тип точности. О нем мы поговорим далее.

TensorRT precision mode

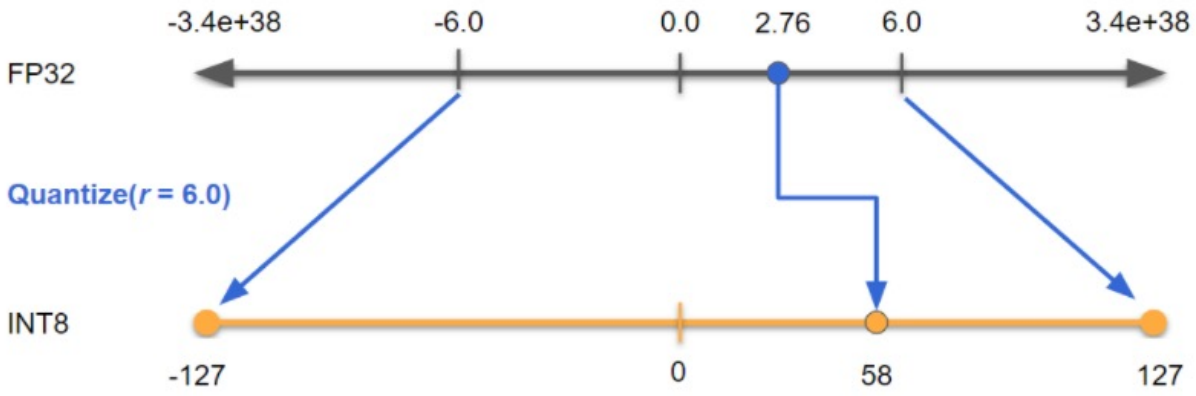
Базовый вариант это FP32, произойдет только оптимизация графа, описанная ранее.

Если вы счастливый обладатель видеокарт поколения от Turing/Volta/Ampere/Hopper, то вам доступны также и точности FP16, INT8. На Pascal тоже будет работать, например, но в производительности ничего не поменяется.

FP16 позволяет снизить точность чисел и тем самым ускорить модель.

TensorRT INT8

INT8 требует уже от нас калибровочного датасета, так как нам нужно откалиброваться на интервал [-127;127]. Основная идея в чем? Веса часто слабо отличаются друг от друга и их можно сгруппировать, используя квантизацию.



$$Quantize(x, r) = round(s * clip(x, -r, r))$$
where $s = 127 / r$

https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html

Потери в точности

Как показывают исследования, сильной потери в точности не будет, все кажется довольно приличным.

Table 1. Verified Models

	Native TensorFlow FP32	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8	
	Volta and Turing	Volta and Turing	Volta and Turing	Volta	Turing
MobileNet v1	71.01	71.01	70.99	69.49	69.49
MobileNet v2	74.08	74.08	74.07	73.96	73.96
NASNet - Large	82.72	82.71	82.70	Work in progress	82.66
NASNet - Mobile	73.97	73.85	73.87	73.19	73.25
ResNet-50 v1.5[1]	76.51	76.51	76.48	76.23	76.23
ResNet-50 v2	76.43	76.37	76.4	76.3	76.3
VGG16	70.89	70.89	70.91	70.84	70.78
VGG19	71.01	71.01	71.01	70.82	70.90
Inception v3	77.99	77.99	77.97	77.92	77.93
Inception v4	80.19	80.19	80.19	80.14	80.08

https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html#performance

TF-TRT

Как мы уже ранее отмечали, TensorRT интегрирован в Tensorflow и его использование заметно упрощается.

1) Обучаем модель на tf или tf.keras
2) Сохраняем модель
3) Загружаем модель
4) Оптимизируем
5) Profit!

Оптимизация resnet50 для классификации imagenet:

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
202 img/sec	465 img/sec	467 img/sec	1939 img/sec

TF-TRT

Результаты зависят от архитектуры, видеокарты и ее характеристик.

Tesla V100 16GB при оптимизации ResNet152V2

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
455 img/sec	565 img/sec	1693 img/sec	1746 img/sec

Tesla V100 16GB при оптимизации ResNet50

Base TF	TF-TRT FP32	TF-TRT FP16	TF-TRT INT8
973 img/sec	1389 img/sec	3874 img/sec	4031 img/sec

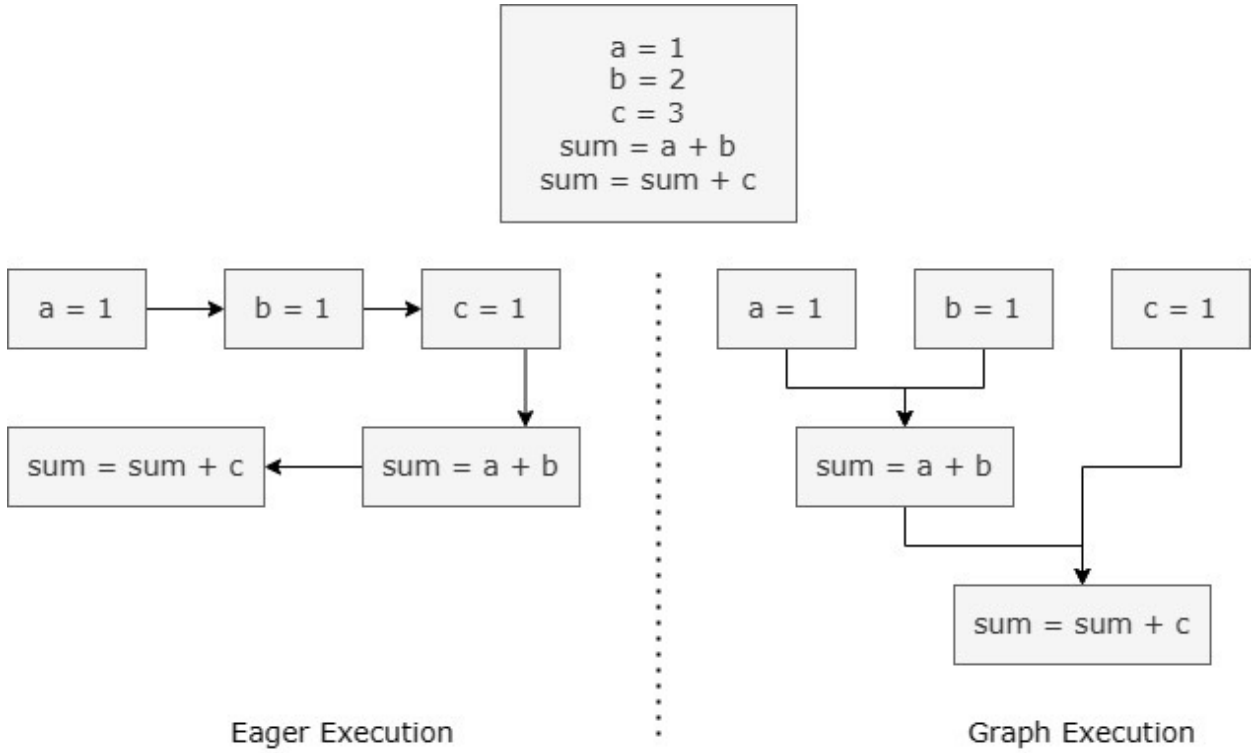
PyTorch

2.0

Eager and Graph Executions

PyTorch

- Выполняем операции в runtime;
- Легко писать, отлаживать;
- Не может до конца использовать весь потенциал GPU.



TensorFlow

- Строим граф всех операций;
- Сложнее писать и отлаживать;
- Оптимизируется и использует все возможности GPU.

PyTorch 2.0

PyTorch всегда стремился к высокой производительности и eager execution. Внутренние компоненты написаны на C++.
Цель ~~1.14~~ 2.0 – поддержание динамических вычислений, ускорение обучения при низком потреблении памяти.

Основа PyTorch 2.0:

- TorchDynamo;
- AOT Autograd;
- PrimTorch;
- TorchInductor.

PyTorch 2.0 backbone

TorchDynamo

Использует JIT-компиляцию для преобразования обычной программы Python в FX граф. FX граф – промежуточное представление кода, которое может быть дополнительно скомпилировано и оптимизировано. TorchDynamo извлекает FX графы проверяя байт-код Python в runtime и находит участки вызова PyTorch.
В любой момент может прервать граф и разить его на части, если увидит вызов, не относящийся к PyTorch.
Стало возможно благодаря PEP 523 – Adding a frame evaluation API to CPython.

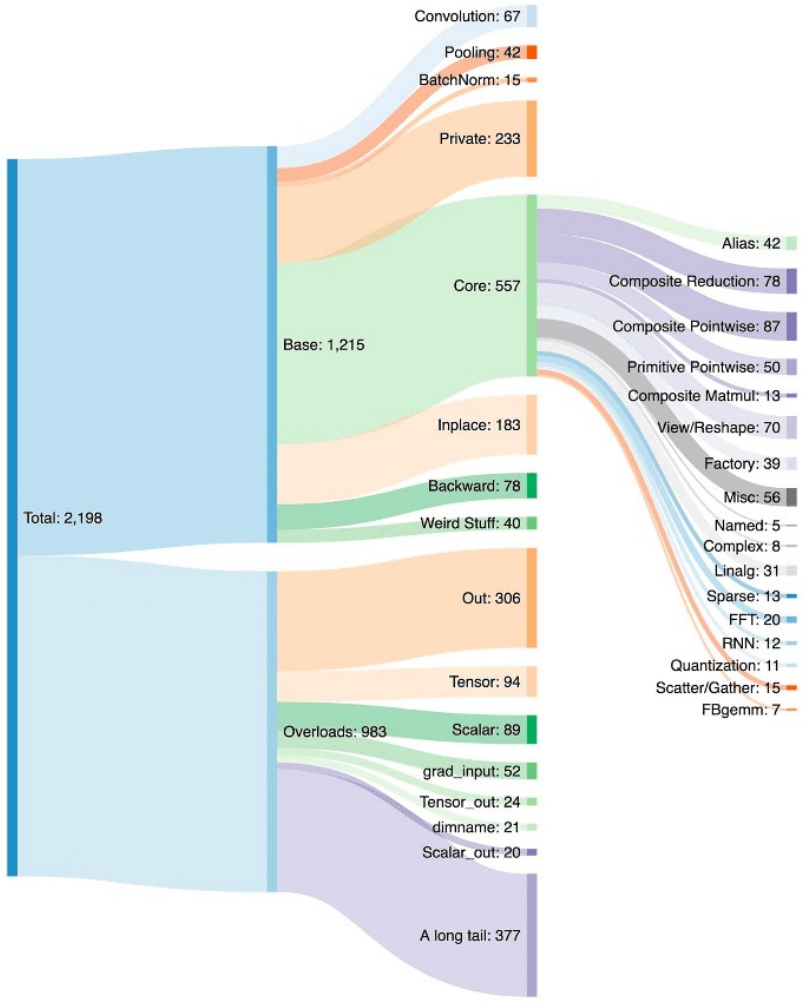
AOT Autograd

Новый механизм автоматического дифференцирования, использует AOT компиляцию для ускорения forward и backward проходов.

TorchInductor

Написанный backend компилятора для компиляции FX графа в машинный код, оптимизированные C++/Triton kernels для CPU/GPU соответственно.

PyTorch 2.0 backbone



PrimTorch

В PyTorch было более 2000 операторов и сложно развивать backend PyTorch. Решение – свести их к более мелким примитивным операциям, по итогу получили 250 примитивов (Primops) и 750 канонических операторов (ATen ops)

torch.compile()

TORCH.COMPILE

torch.compile(*model=None*, ***, *fullgraph=False*, *dynamic=False*, *backend='inductor'*, *mode=None*, *options=None*, *disable=False*) [SOURCE]

Optimizes given model/function using TorchDynamo and specified backend.

Parameters:

- **model** (*Callable*) – Module/function to optimize
- **fullgraph** (*bool*) – Whether it is ok to break model into several subgraphs
- **dynamic** (*bool*) – Use dynamic shape tracing
- **backend** (*str or Callable*) – backend to be used
- **mode** (*str*) – Can be either “default”, “reduce-overhead” or “max-autotune”
- **options** (*dict*) – A dictionary of options to pass to the backend.
- **disable** (*bool*) – Turn torch.compile() into a no-op for testing

torch.compile()

model – экземпляр класса nn.Module

mode – отвечает за процесс компиляции:

- default – эффективная компиляция без высоких временных затрат;
- reduce-overhead - значительно сокращает накладные расходы, но потребляет небольшой объем дополнительной памяти;
- max-autotune – на выходе самый быстрый код, но тратит много времени.

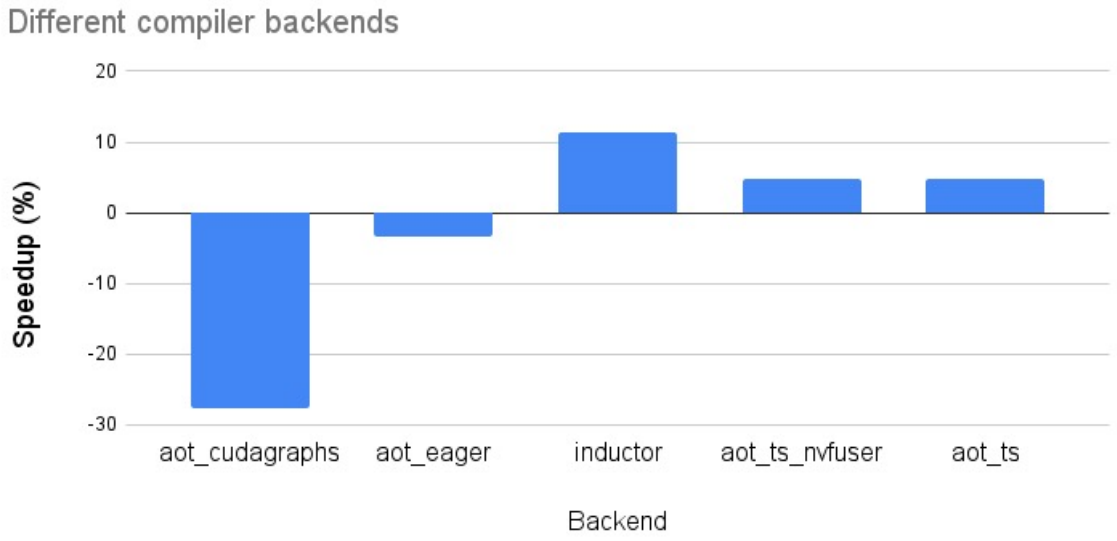
Default Mode	**Reduce Overhead Mode**	**Max Autotune Mode**
Optimizes for large models	Optimizes for small models	Optimizes to produce the fastest models
Low compile time	Low compile time	Very high compile time
No extra memory usage	Uses some extra memory	—

torch.compile()

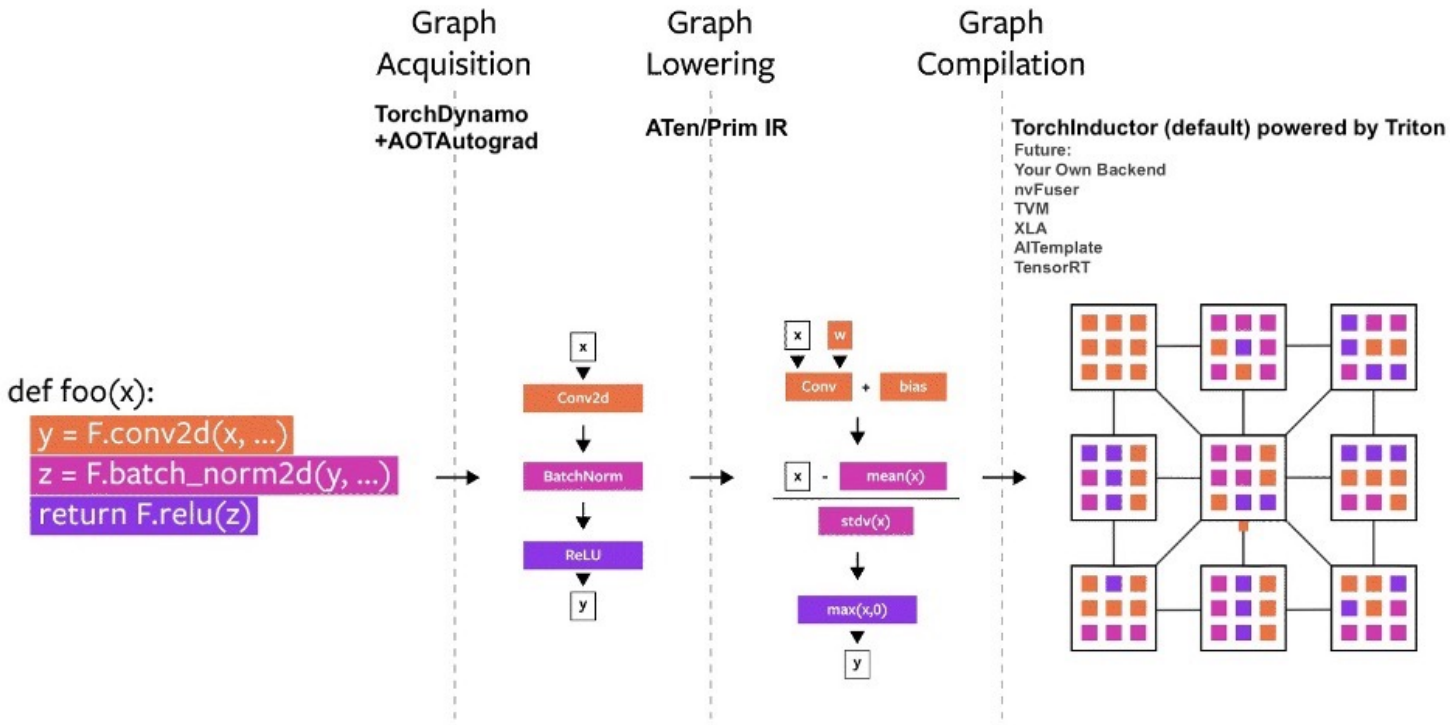
dynamic – до компиляции нужно указать, есть ли у вас элементы динамического графа.

fullgraph – в большинстве случаев False, тогда программа будет разбиваться на подграфы, иначе будет строить единый граф в случае, когда производительность крайне важна.

backend - какой backend использовать для компиляции. По умолчанию TorchInductor, доступны другие, например aot_cudagraphs, nvfuser.



torch.compile()



Разбиваем модель на подграфы, что-то оптимизируем, что-то возвращаем в eager.

Декомпозируем все операции в выбранные backend-specific kernels

Все backend kernels вызывают соответствующие им низкоуровневые операции

https://pytorch.org/get-started/pytorch-2.0/#pytorch-2x-faster-more-pythonic-and-as-dynamic-as-ever
По ссылке больше информации