A Hugging Face Accelerate Story of Multiple Backends: FSDP and DeepSpeed

Published June 13, 2024

mirinflim
Yu Chin Fabian
guest

aldopareja
aldo pareja
guest

muellerzr
Zachary Mueller

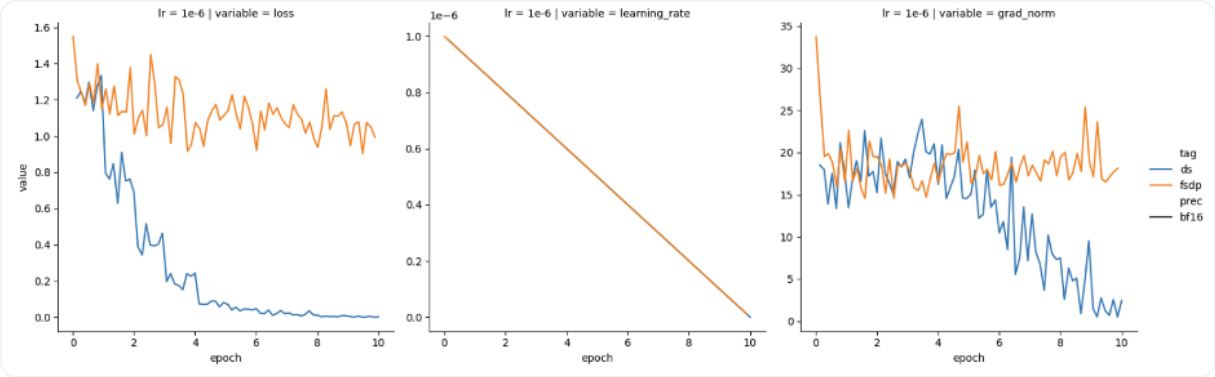
stas
Stas Bekman
ContextualAI

There are two popular implementations of the ZeRO Redundancy Optimizer (Zero) algorithm in the community, one from DeepSpeed and the other from PyTorch. Hugging Face Accelerate exposes both these frameworks for the end users to train/tune their models. This blog highlights the differences between how these backends are exposed through Accelerate. To enable users to seamlessly switch between these backends, we upstreamed a precision-related change and a concept guide.

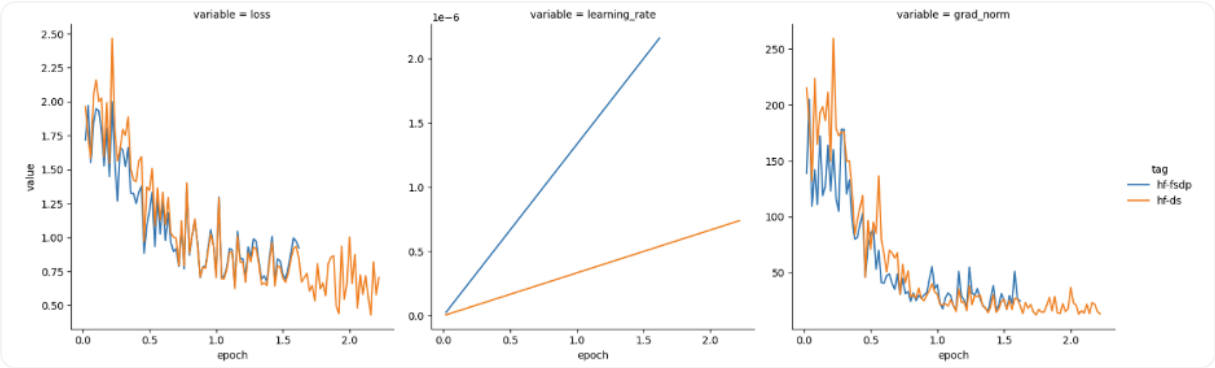
Are FSDP and DeepSpeed Interchangeable?

Recently, we tried running a training pipeline with DeepSpeed and PyTorch FSDP. We noticed that the results obtained differed. The specific model was Mistral-7B

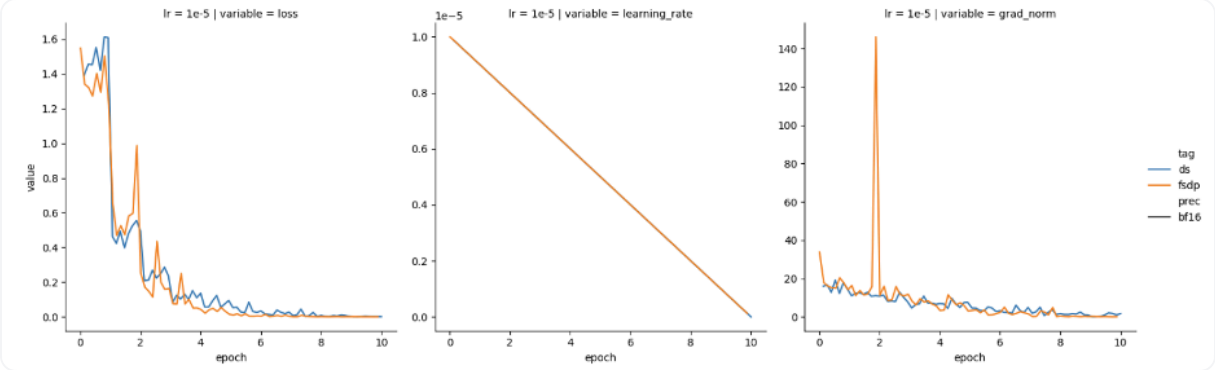
base and it was loaded in half-precision (`bfloat16`). While the DeepSpeed (blue) loss had converged well, the FSDP (orange) loss was not decreasing, as can be seen in Figure 1.



We hypothesized that the learning rate may need scaling by the number of GPUs and bumped up the learning rate by 4x since we were using 4 GPUs. Then, we saw the following loss behavior, shown in Figure 2.



It looked like the desired behavior had been achieved by scaling the FSDP learning rate by the number of GPUs! However, when we tried a different learning rate (`1e-5`) without scaling, we observed similar loss and gradient norm characteristics for both frameworks, shown in Figure 3.



Precision Matters

Inside the `DeepSpeed` codebase, specifically in the implementation of `DeepSpeedZeroOptimizer_Stage3` (as the name implies, what handles doing Stage 3 optimizer sharding), we noticed that the `trainable_param_groups`, the parameter groups being trained on, pass through an internal `_setup_for_real_optimizer` function call, which calls another function called `_create_fp32_partitions`. As the `fp32` in the name suggests, `DeepSpeed` was performing upcasting internally, and it always keeps its master weights in `fp32` by design. This upcasting to full precision meant that the optimizer could converge at learning rates that it would not converge in lower precision. The earlier observations were artifacts of this precision difference.

In FSDP, before the model and optimizer parameters are distributed across GPUs, they are first "flattened" to a one-dimensional tensor. FSDP and DeepSpeed use different `dtype`s for these "flattened" parameters which has ramifications for PyTorch optimizers. Table 1 outlines the processes for both frameworks; the "Local" column indicates the process occurring per-GPU, therefore the memory overhead from upcasting is amortized by the number of GPUs.

Process	Local?	Framework	Details
Loading the model in (such as `AutoModel.from_pretrained(...,`	❌		

Process	Local?	Framework	Details
`torch_dtype=torch_dtype))`			
Preparation, such as creation of the "flattened parameters"	✅	FSDP DeepSpeed	utilizes `torch_dtype` disregards `torch_dtype` and is created in `float32`
Optimizer initialization	✅	FSDP DeepSpeed	creates parameters in `torch_dtype` creates parameters in `float32`
Training Step (forward, backward, reduction)	❌	FSDP DeepSpeed	follows fsdp.MixedPrecision follows `deepspeed_config_file` mixed precision settings
Optimizer (pre-step)	✅	FSDP DeepSpeed	upcasting (if any) to `torch_dtype` upcasting everything to `float32`
Optimizer (actual step)	✅	FSDP DeepSpeed	occurs in `torch_dtype` occurs in `float32`

> *"Table 1: Summary of how FSDP and DeepSpeed handle mixed precision"*

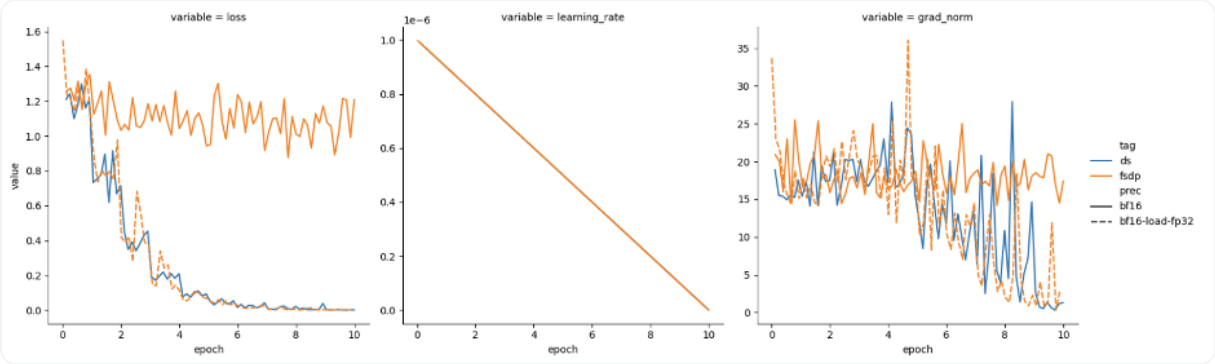
A few takeaway points:

- As noted an 🤗 Accelerate issue, a rule of thumb when performing mixed precision is to keep trainable parameters in `float32`.
- Upcasting, as is done in `DeepSpeed`, may have a negligible effect on memory consumption when sharding over a large number of GPUs. However, when using `DeepSpeed` on a small number of GPUs, the 2x increase in memory consumption can be significant.

- The torch-native implementation of FSDP does not force upcasting, allowing a user to operate PyTorch optimizers in low precision. This offers more flexibility than the native upcasting of `DeepSpeed`.

Harmonizing DeepSpeed and FSDP in 🤗 Accelerate

To better align DeepSpeed and FSDP in 🤗 Accelerate, we can perform upcasting automatically for FSDP when mixed precision is enabled. We created a pull request with this change that was included in the 0.30.0 release.



The result of this PR is to allow FSDP to operate in two modes:

- A “mixed-precision” mode like the DeepSpeed counterpart
- A low precision mode for memory constrained scenarios, as shown in Figure 4.

The two new FSDP modes are summarized in Table 2 and compared with DeepSpeed.

Framework	Model Loading (`torch_dtype`)	Mixed Precision	Preparation (Local)	Training	Optimizer (Local)
FSDP (memory-constrained)	`bf16`	default (none)	`bf16`	`bf16`	`bf16`

Framework	Model Loading (`torch_dtype`)	Mixed Precision	Preparation (Local)	Training	Optimizer (Local)
FSDP (mixed precision mode)	`bf16`	`bf16`	`fp32`	`bf16`	`fp32`
DeepSpeed	`bf16`	`bf16`	`fp32`	`bf16`	`fp32`

> *"Table 2: Summary of the two new FSDP modes and comparisons with DeepSpeed"*

Throughput results

We use the IBM Granite 7B model (which follows the Meta Llama2 architecture) for throughput comparisons. We compare Model Flops Utilization (MFU) and tokens/sec/GPU metrics and show them for FSDP (full sharding) and DeepSpeed (Zero3).

We used four A100 GPUs as before with the following hyperparameters:

- Batch size of 8
- Model loaded in `torch.bfloat16`
- Mixed precision is the same dtype.

Table 3 shows that FSDP and DeepSpeed are expected to perform similarly.

> *"We intend to follow up with a comprehensive throughput comparison and approaches to improve throughput (e.g., 4D masks with packing, torch.compile, selective activation checkpointing) as large scale alignment techniques like InstructLab and GLAN become popular."*

Framework	Tokens / sec / device	Step time (s)	Model Flops Utilization (MFU)
FSDP (aligned mode)	3158.7	10.4	0.41
DeepSpeed	3094.5	10.6	0.40

> *"Table 3: Ballpark throughput comparisons between FSDP and DeepSpeed on four A100 GPUs."*

Closing thoughts

We provided a new concept guide to help users migrate between the two frameworks. The guide helps users answer questions such as:

- How do we achieve equivalent sharding strategies?
- How do we perform efficient model loading?
- How is weight prefetching managed in FSDP and DeepSpeed?
- What is the equivalent of FSDP wrapping in DeepSpeed?

We consider various modes of configuring these frameworks in 🤗 Accelerate,

- From the command line during `accelerate launch`
- From the various `Plugin` classes 🤗 Accelerate provides for (`DeepSpeed`) [https://huggingface.co/docs/accelerate/main/en/package_reference/deepsp and (`FSDP`) [https://huggingface.co/docs/accelerate/main/en/package_reference/fsdp]

🤗 Accelerate makes it almost **trivial** to switch between FSDP and DeepSpeed, with the majority of it being an Accelerate config file change (see the new concept guide

for instructions on this).

Besides the config change, some of the other considerations (also outlined in the guide) are differences in how checkpoints are handled, etc.

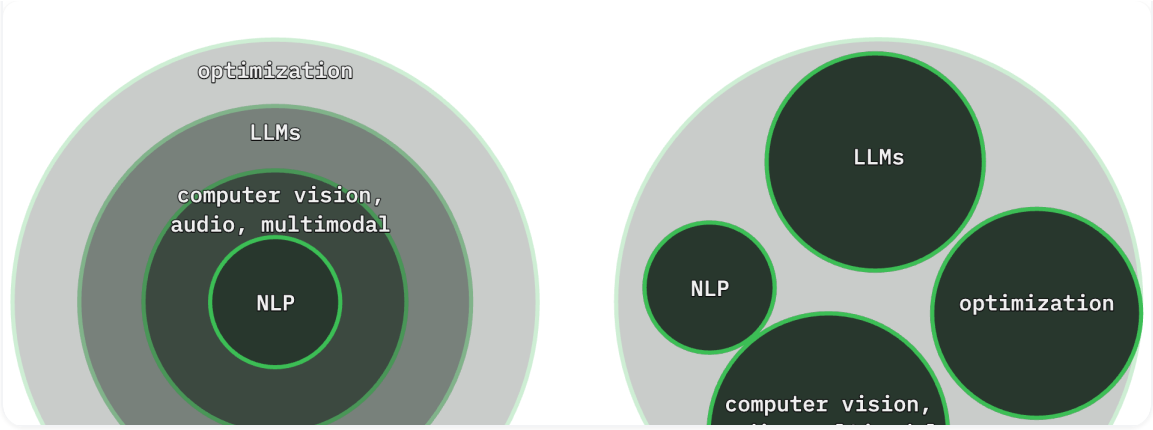
All experiments in this blog can be reproduced with the code from the original 🤗 Accelerate issue.

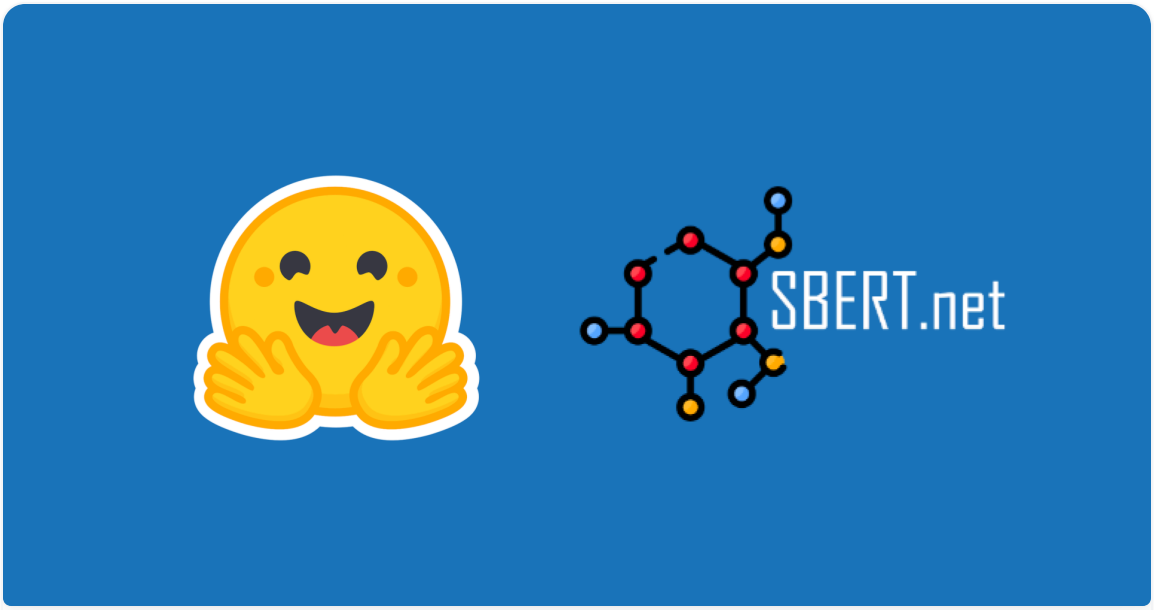
We intend to follow up with throughput comparisons at scale and techniques to better utilize those GPUs for tuning and alignment jobs while maintaining model quality.

Acknowledgements

This is an effort that involved several teams across multiple organizations to come together. It started at IBM Research, specifically Aldo Pareja, who found the issue, and Fabian Lim, who identified the precision gaps and fixed this issue. Zach Mueller and Stas Bekman have been phenomenal in providing feedback and the fixes to accelerate. Less Wright from the PyTorch Team at Meta was very helpful with questions on FSDP parameters. Finally, we would also like to thank the DeepSpeed team for providing feedback on this blog.

More Articles from our Blog





Training and Finetuning Embedding Models with Sentence Transformers v3

By tomaarsen May 27, 2024 • △ 94

Company

TOS