DeepSpeed

Release 0.3.0

Nov 25, 2020

Contents

1 Model Setup **1**
1.1 Training Setup 1

2 Training API **3**
2.1 Training API 3

3 Checkpointing API **5**
3.1 Model Checkpointing 5
3.2 Activation Checkpointing 5

4 Transformer Kernel API **7**
4.1 Transformer Kernels 7

5 Pipeline Parallelism **9**
5.1 Pipeline Parallelism 9

6 Indices and tables **11**

CHAPTER 1

Model Setup

1.1 Training Setup

1.1.1 Argument Parsing

DeepSpeed uses the argparse library to supply commandline configuration to the DeepSpeed runtime. Use `deepspeed.add_config_arguments()` to add DeepSpeed's builtin arguments to your application's parser.

```
parser = argparse.ArgumentParser(description='My training script.')
parser.add_argument('--local_rank', type=int, default=-1,
                    help='local rank passed from distributed launcher')
# Include DeepSpeed configuration arguments
parser = deepspeed.add_config_arguments(parser)
cmd_args = parser.parse_args()
```

1.1.2 Training Initialization

The entrypoint for all training with DeepSpeed is `deepspeed.initialize()`.

Example usage:

```
model_engine, optimizer, _, _ = deepspeed.initialize(args=cmd_args,
                                                     model=net,
                                                     model_parameters=net.
↪parameters())
```

CHAPTER 2

Training API

2.1 Training API

`deepspeed.initialize()` returns a *training engine* in its first argument of type `DeepSpeedEngine`. This engine is used to progress training:

```
for step, batch in enumerate(data_loader):
    #forward() method
    loss = model_engine(batch)

    #runs backpropagation
    model_engine.backward(loss)

    #weight update
    model_engine.step()
```

2.1.1 Forward Propagation

2.1.2 Backward Propagation

2.1.3 Optimizer Step

2.1.4 Gradient Accumulation

CHAPTER 3

Checkpointing API

3.1 Model Checkpointing

DeepSpeed provides routines for checkpointing model state during training.

3.1.1 Loading Training Checkpoints

3.1.2 Saving Training Checkpoints

3.2 Activation Checkpointing

The activation checkpointing API's in DeepSpeed can be used to enable a range of memory optimizations relating to activation checkpointing. These include activation partitioning across GPUs when using model parallelism, CPU checkpointing, contiguous memory optimizations, etc.

Please see the DeepSpeed JSON config for the full set.

Here we present the activation checkpointing API. Please see the enabling DeepSpeed for Megatron-LM tutorial for example usage.

3.2.1 Configuring Activation Checkpointing

3.2.2 Using Activation Checkpointing

3.2.3 Configuring and Checkpointing Random Seeds

CHAPTER 4

Transformer Kernel API

4.1 Transformer Kernels

The transformer kernel API in DeepSpeed can be used to create BERT transformer layer for more efficient pre-training and fine-tuning, it includes the transformer layer configurations and transformer layer module initialization.

Here we present the transformer kernel API. Please see the BERT pre-training tutorial for usage details.

4.1.1 DeepSpeed Transformer Config

4.1.2 DeepSpeed Transformer Layer

CHAPTER 5

Pipeline Parallelism

5.1 Pipeline Parallelism

5.1.1 Model Specification

5.1.2 Training

5.1.3 Extending Pipeline Parallelism

CHAPTER 6

Indices and tables

- genindex
- modindex
- search