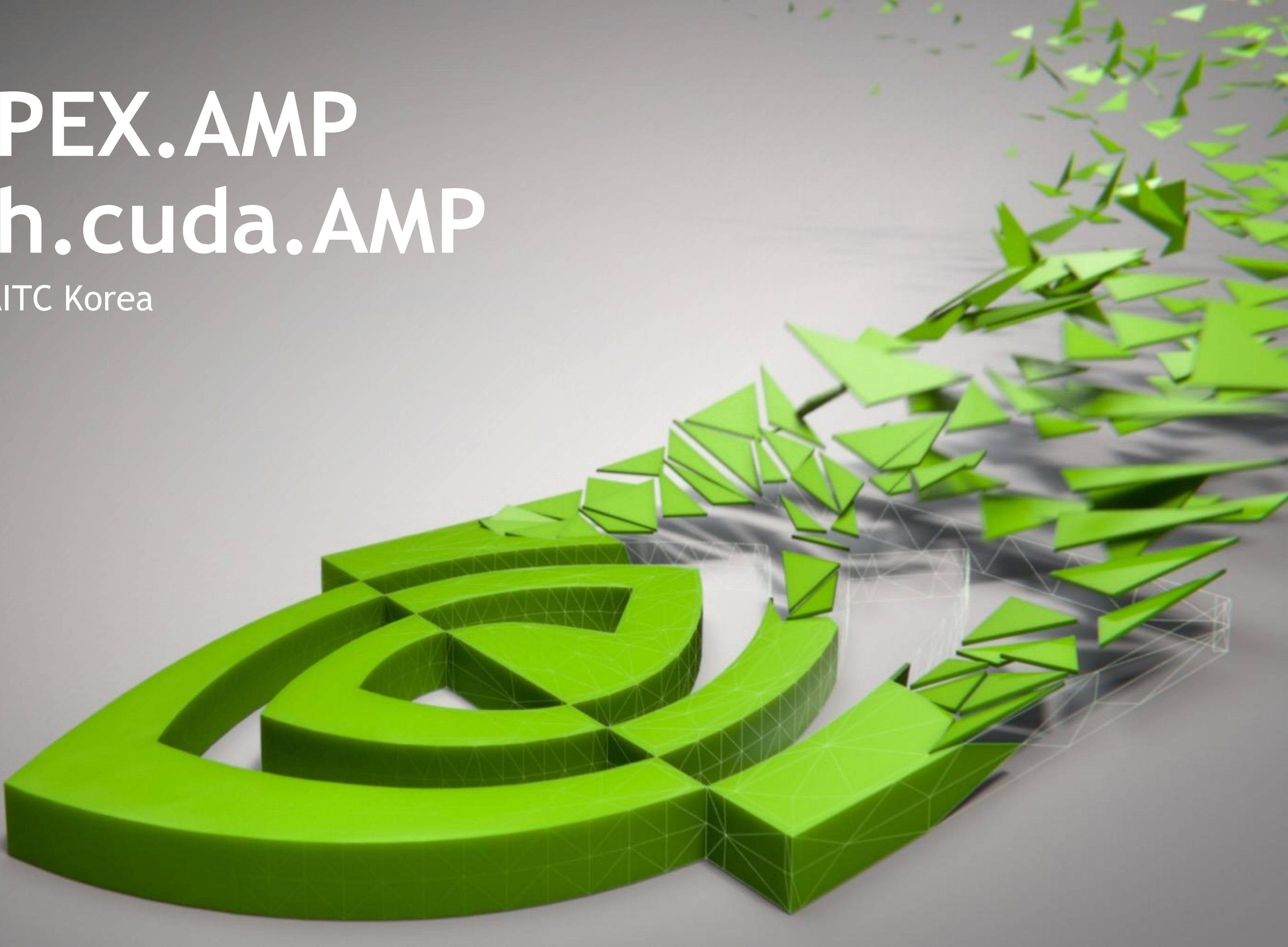
From APEX.AMP To torch.cuda.AMP

Hyungon Ryu | NVAITC Korea



AGENDA

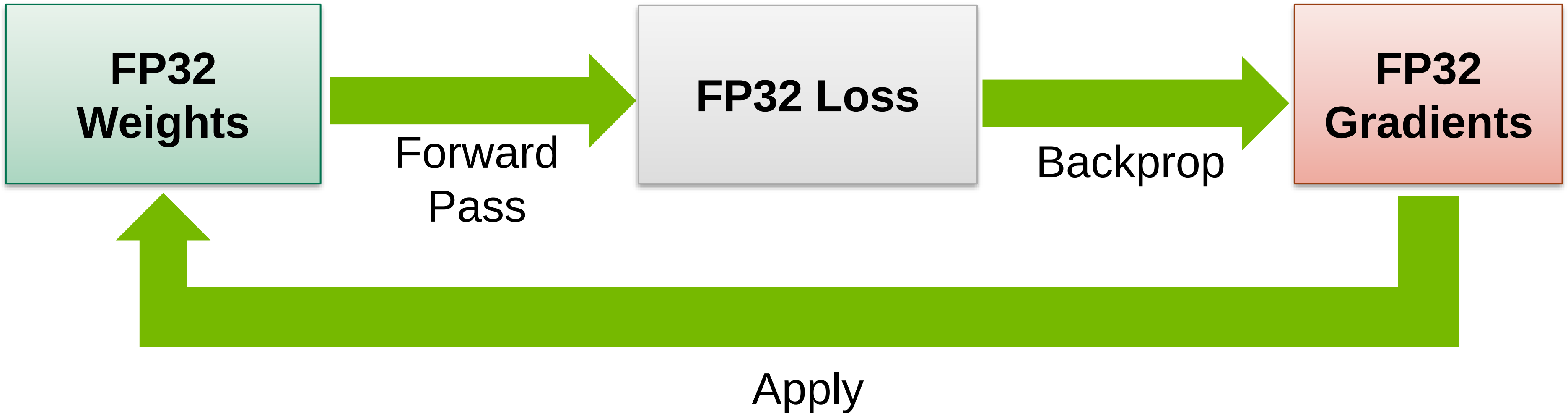
Precision

apex.amp

torch.cuda.amp

PRECISION

FP32 TRAINING



FP32 Weights

Forward Pass

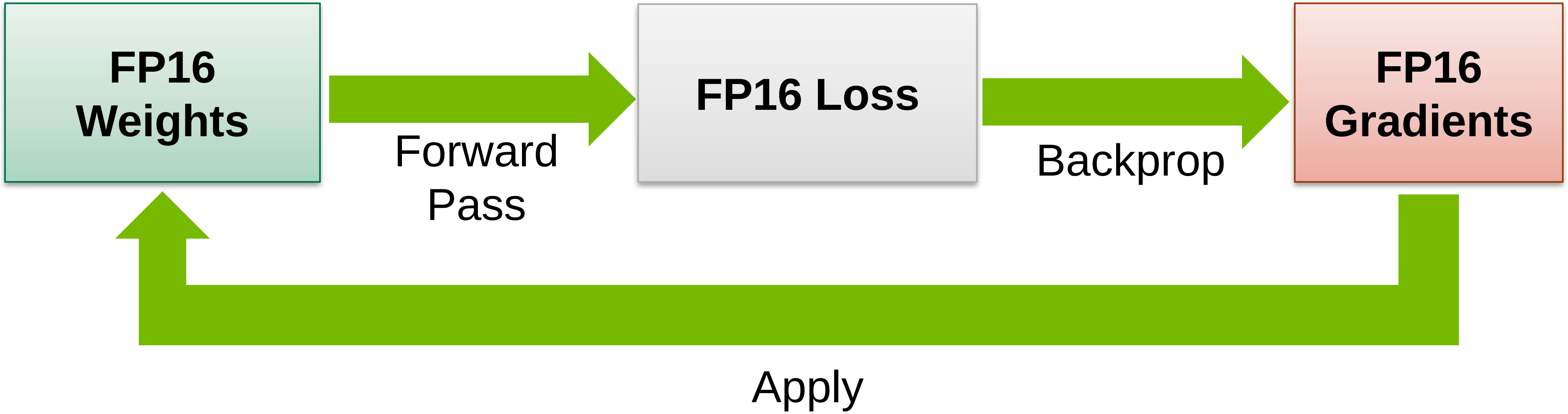
FP32 Loss

Backprop

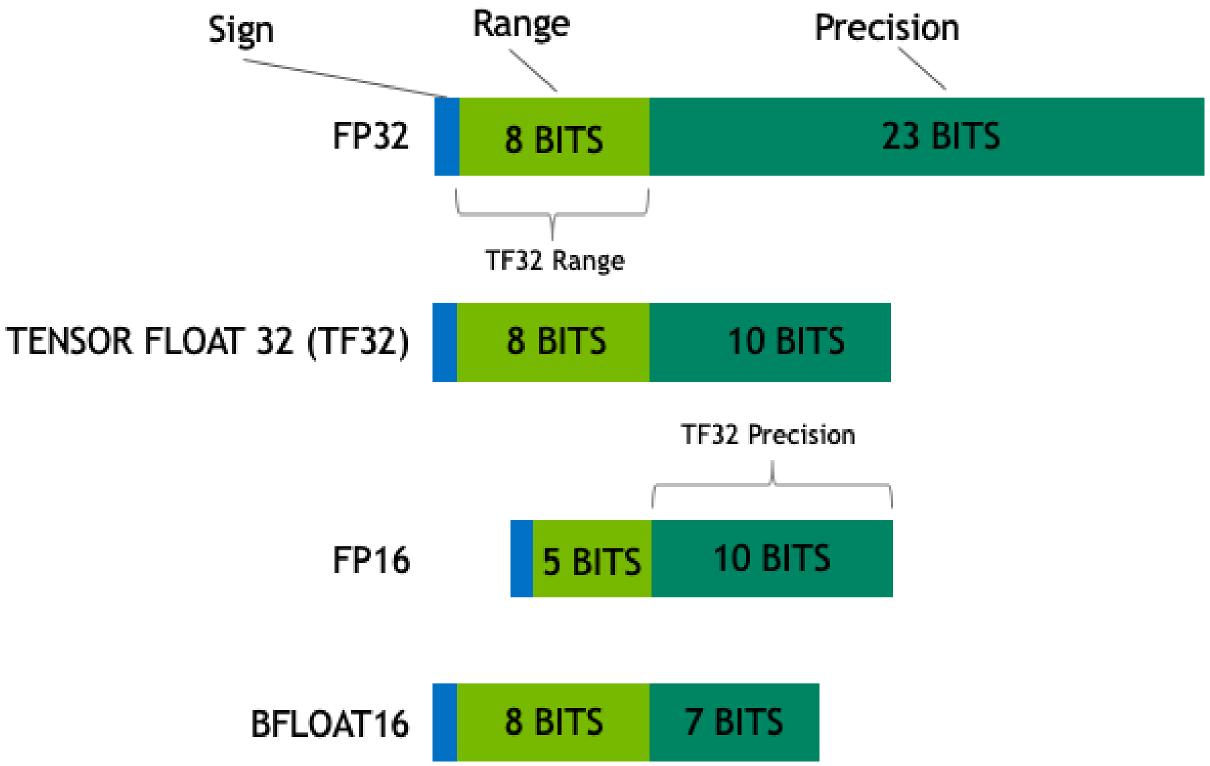
FP32 Gradients

Apply

FP16 TRAINING



PRECISION FOR DL



32Bit

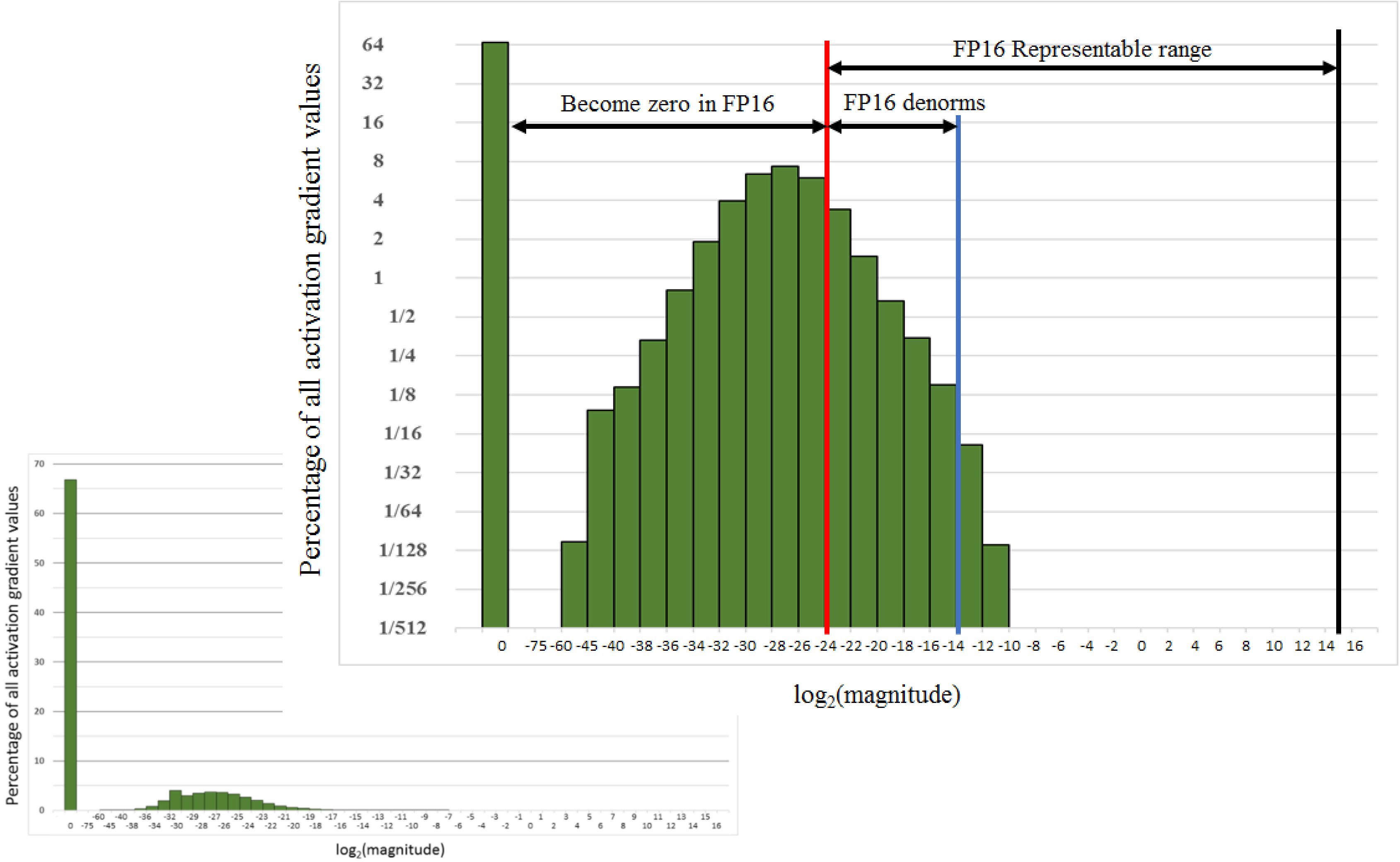
19Bit

16Bit

Benefit of FP16
Lowering required memory
~ large mini-batches
~ less data transfer
Faster training/inferencing

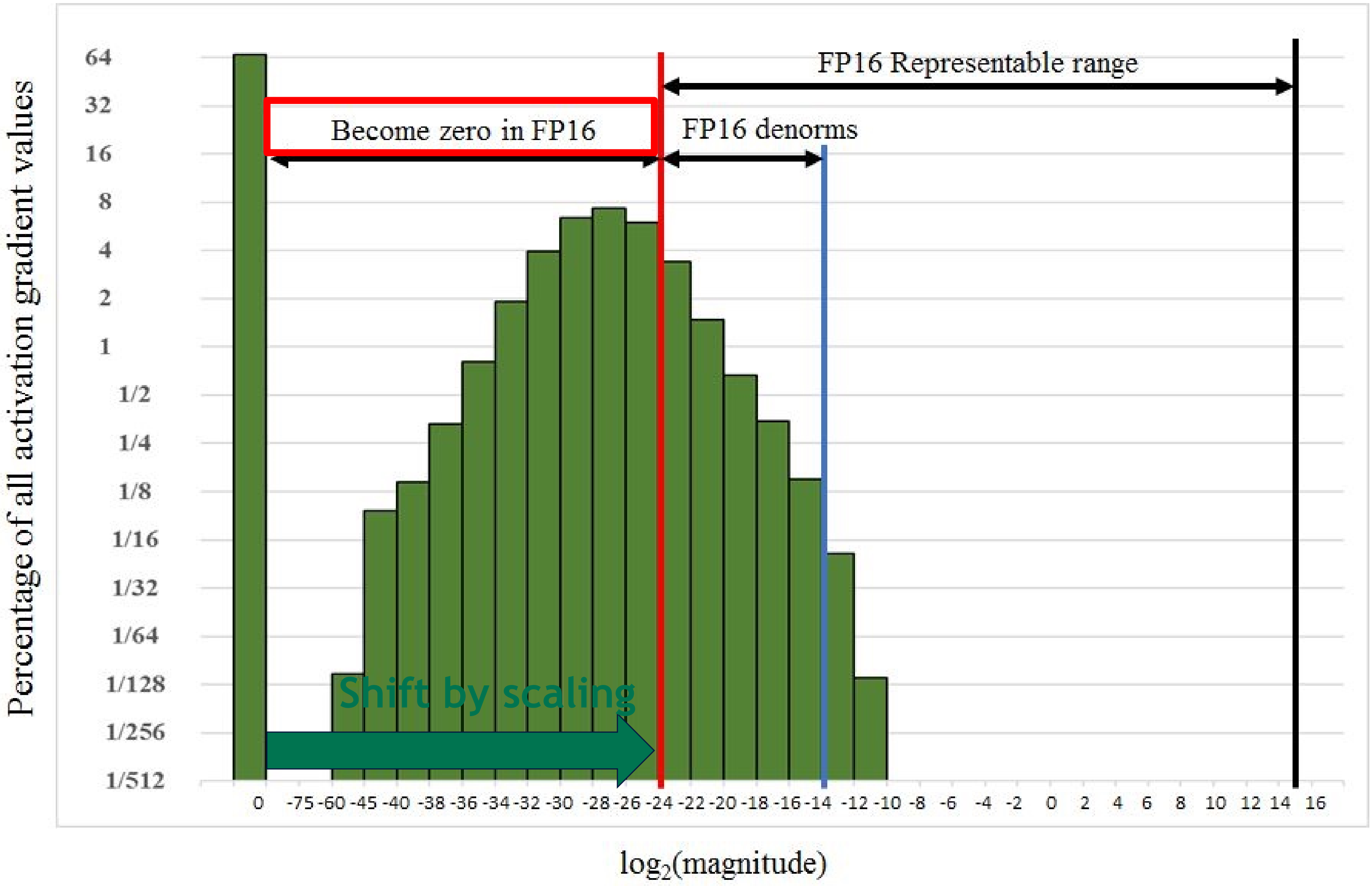
A HISTOGRAM OF ACTIVATION GRADIENTS

Representation of Half Precision floating point format



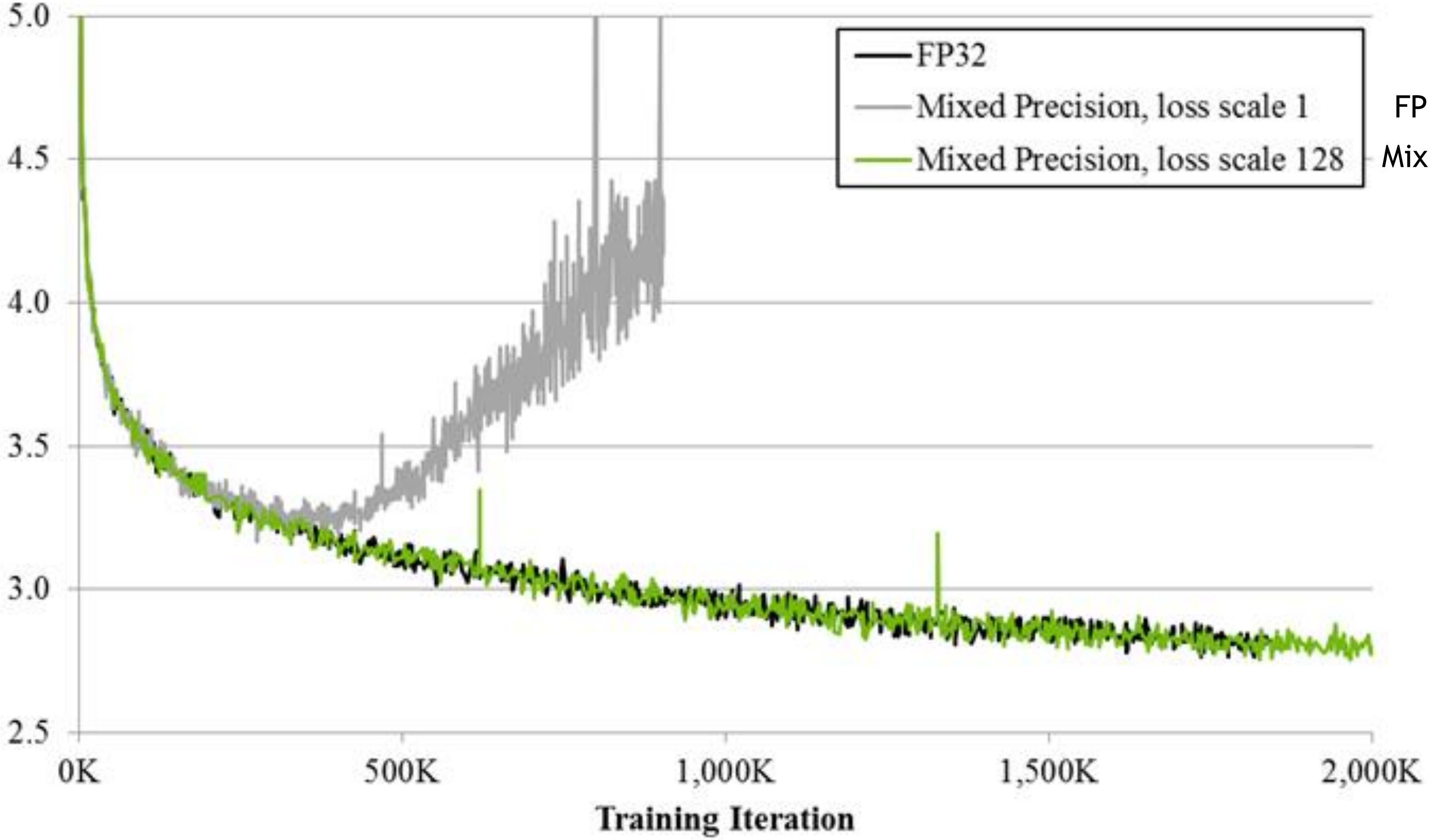
FP32		FP16
0.0000001	→	0
Near Zero		Zero

GRADIENTS MAY UNDERFLOW



https://devblogs.nvidia.com/mixed-precision-training-deep-neural-networks/

GRADIENT EXPLODE

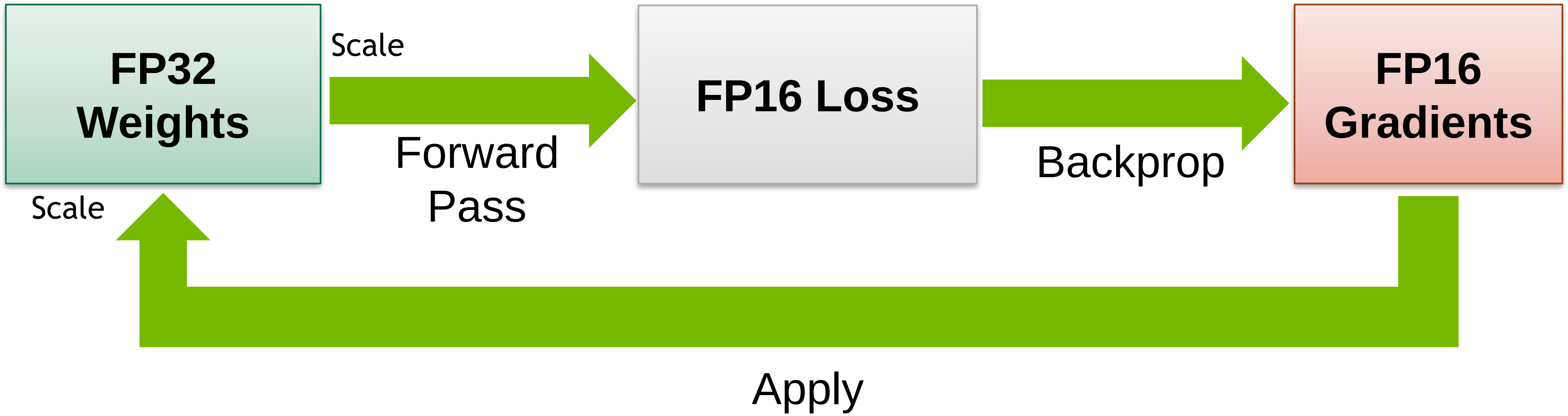


FP16

Mixed Precision

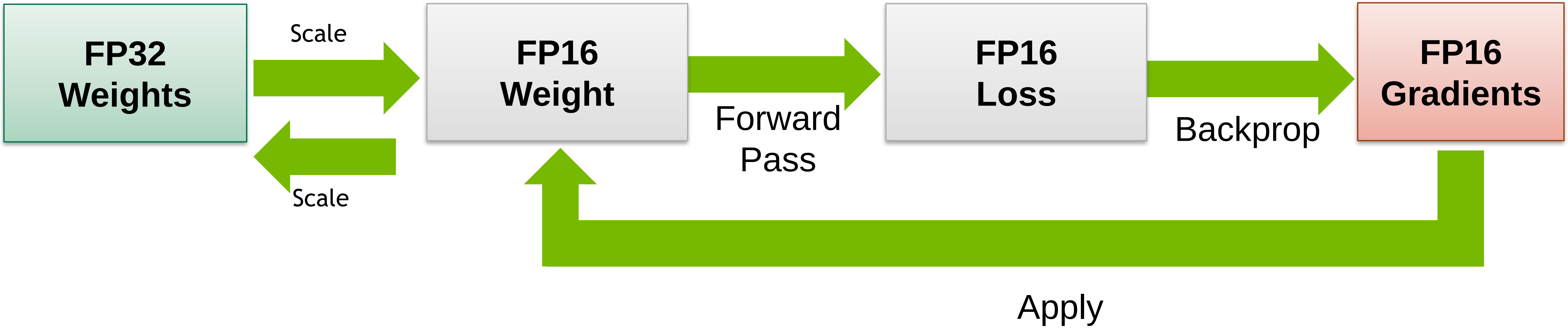
MIXED PRECISION TRAINING

Compute in FP16
Save in FP32



MIXED PRECISION TRAINING

Compute in FP16
Save in FP32



cast to fp16
x loss_scale

cast to fp32
÷loss_scale

APEX.AMP

APEX.AMP

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

APEX enable DDP, Mixed Precision(FP16) and AMP

```
amp.initialize(model, optimizer, opt_level="O1")
```

O0 : FP32, O1/O2 : AMP, O3 : FP16

```
with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```

3 STEP FOR APEX.AMP

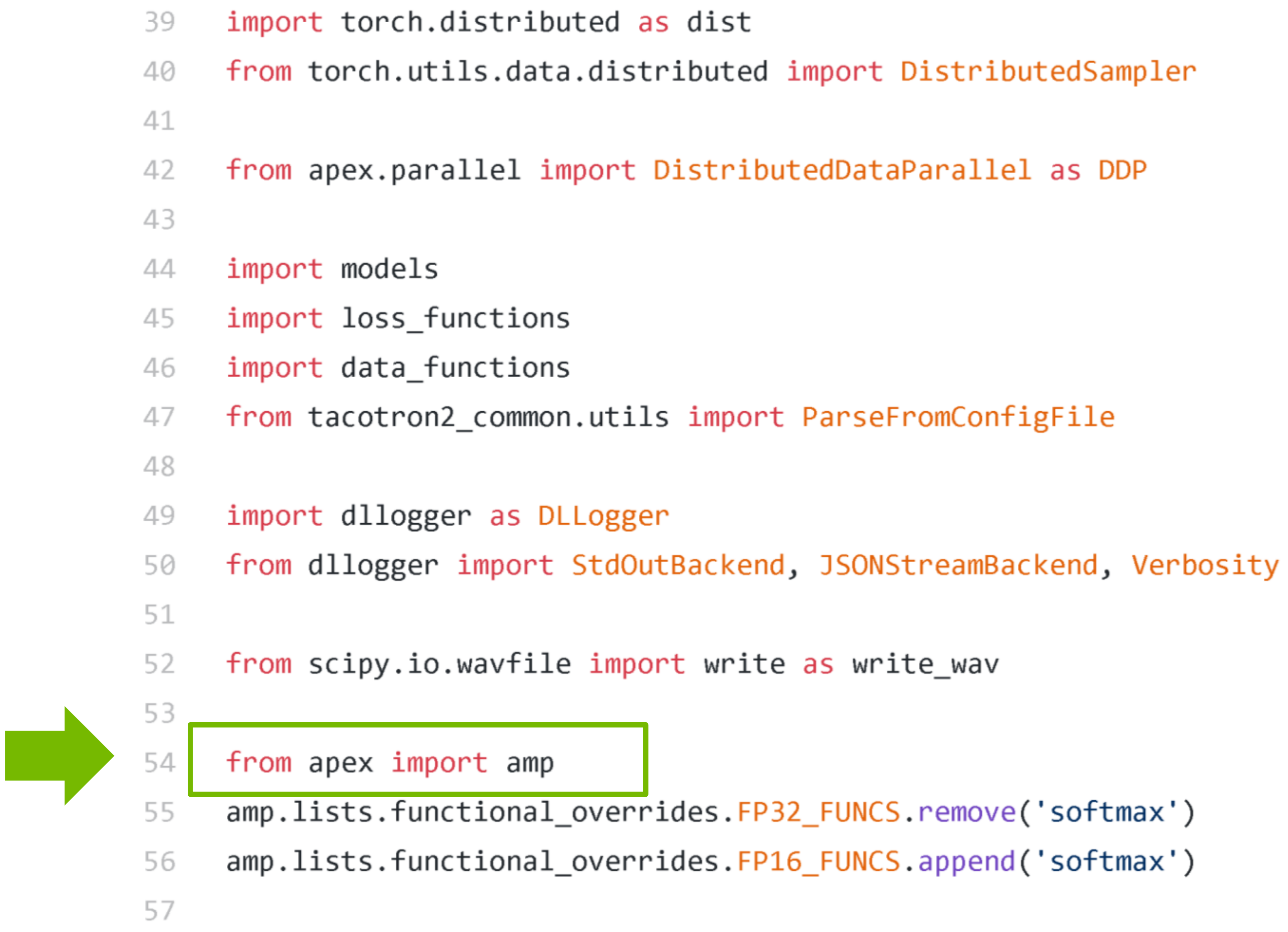
Load APEX.AMP module

Initiate APEX AMP

Enable Loss scale

APEX.AMP

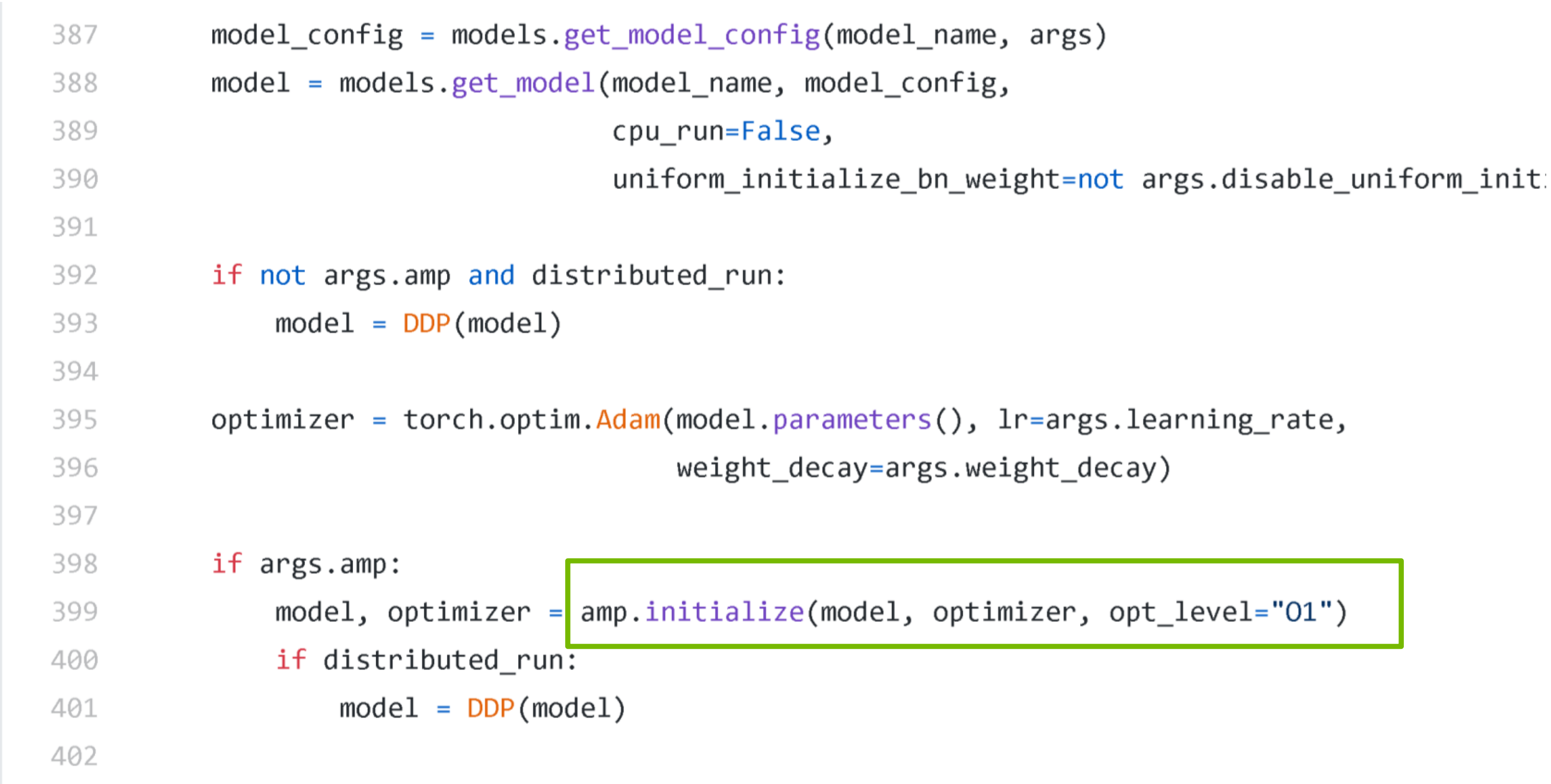
Step1. load APEX module



```
import torch.distributed as dist
from torch.utils.data.distributed import DistributedSampler

from apex.parallel import DistributedDataParallel as DDP

import models
import loss_functions
import data_functions
from tacotron2_common.utils import ParseFromConfigFile

import dllogger as DLLogger
from dllogger import StdOutBackend, JSONStreamBackend, Verbosity

from scipy.io.wavfile import write as write_wav

from apex import amp
amp.lists.functional_overrides.FP32_FUNCS.remove('softmax')
amp.lists.functional_overrides.FP16_FUNCS.append('softmax')

```

APEX.AMP

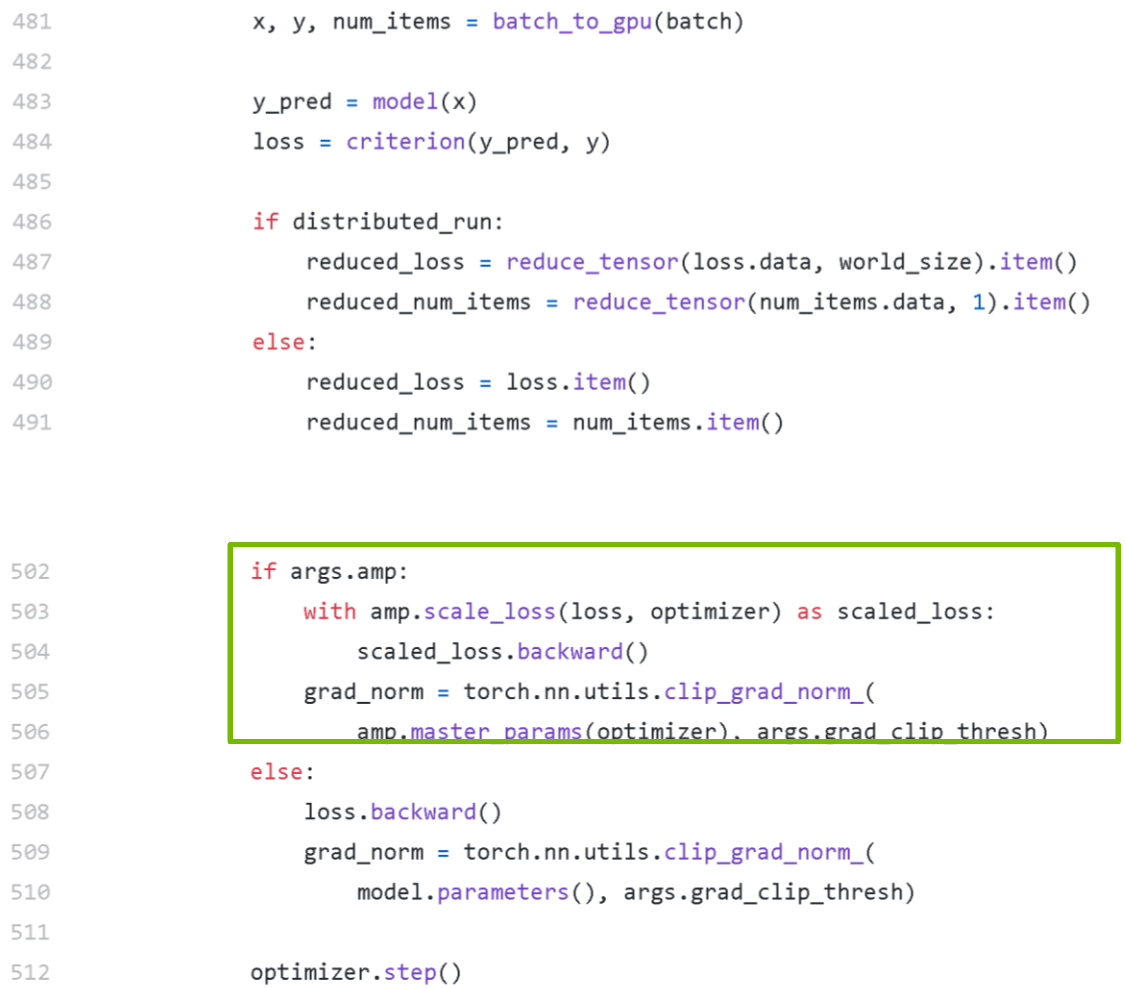
Step2. initiate AMP



```
model_config = models.get_model_config(model_name, args)
model = models.get_model(model_name, model_config,
                         cpu_run=False,
                         uniform_initialize_bn_weight=not args.disable_uniform_init

if not args.amp and distributed_run:
    model = DDP(model)

optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate,
                             weight_decay=args.weight_decay)

if args.amp:
    model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
    if distributed_run:
        model = DDP(model)
```

APEX.AMP

Step3. enable loss scale

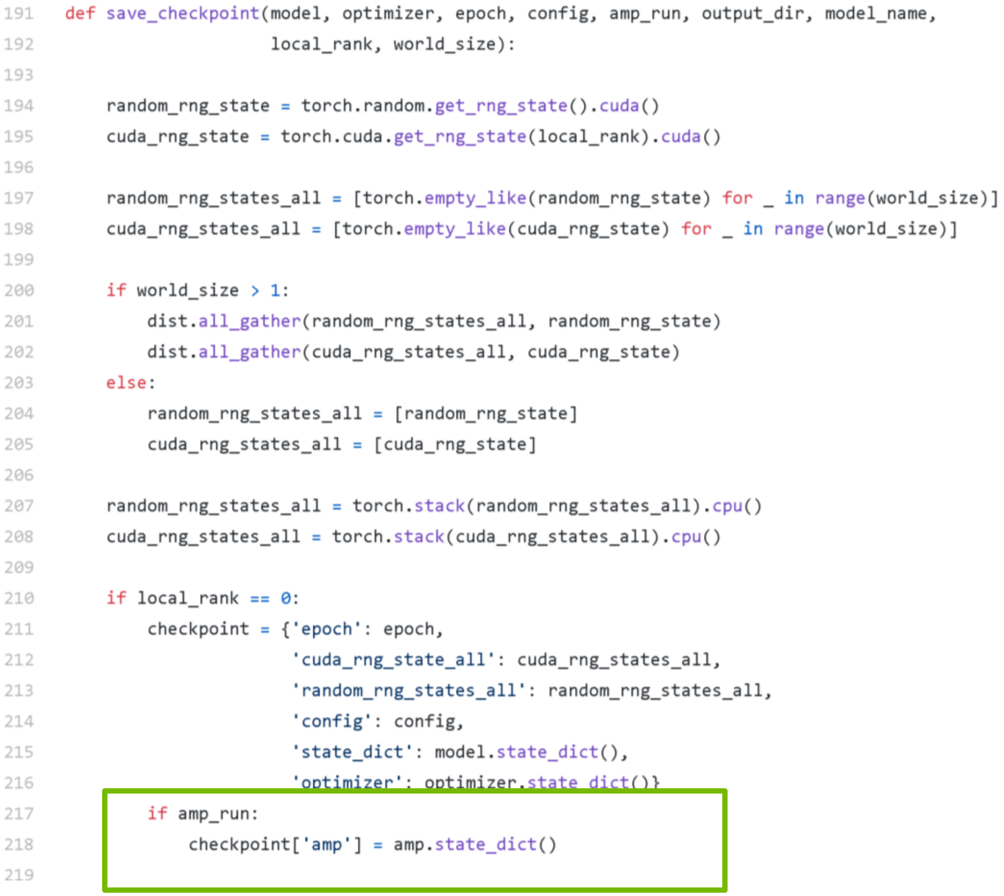


```
x, y, num_items = batch_to_gpu(batch)

y_pred = model(x)
loss = criterion(y_pred, y)

if distributed_run:
    reduced_loss = reduce_tensor(loss.data, world_size).item()
    reduced_num_items = reduce_tensor(num_items.data, 1).item()
else:
    reduced_loss = loss.item()
    reduced_num_items = num_items.item()
```

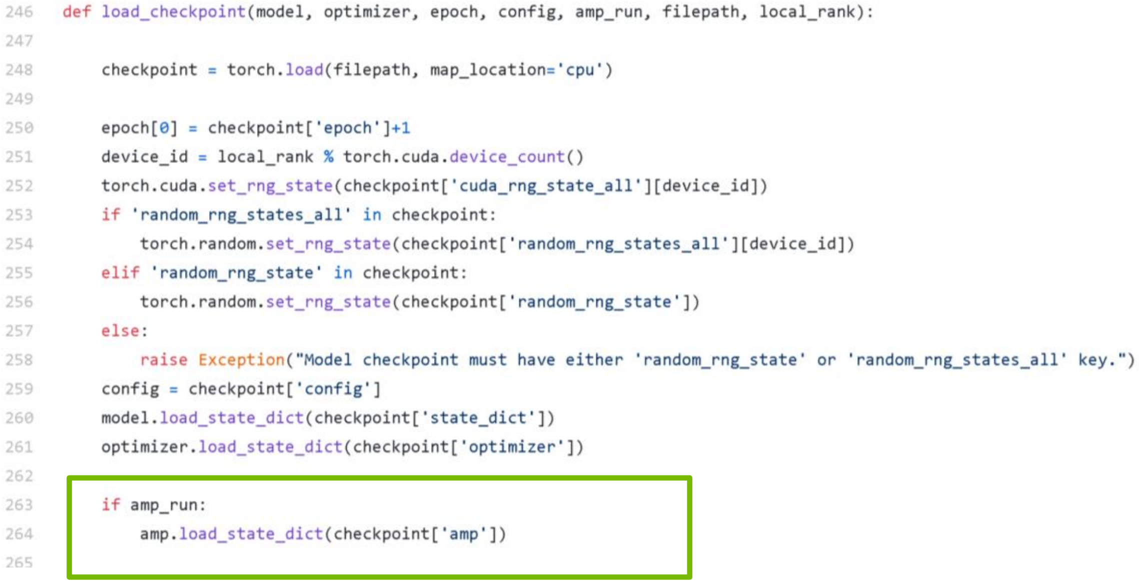
```
if args.amp:
    with amp.scale_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
    grad_norm = torch.nn.utils.clip_grad_norm_(
        amp.master_params(optimizer), args.grad_clip_thresh)
else:
    loss.backward()
    grad_norm = torch.nn.utils.clip_grad_norm_(
        model.parameters(), args.grad_clip_thresh)

optimizer.step()
```

ALSO NEED

amp.state_dict()



```python
def save_checkpoint(model, optimizer, epoch, config, amp_run, output_dir, model_name,
                    local_rank, world_size):

    random_rng_state = torch.random.get_rng_state().cuda()
    cuda_rng_state = torch.cuda.get_rng_state(local_rank).cuda()

    random_rng_states_all = [torch.empty_like(random_rng_state) for _ in range(world_size)]
    cuda_rng_states_all = [torch.empty_like(cuda_rng_state) for _ in range(world_size)]

    if world_size > 1:
        dist.all_gather(random_rng_states_all, random_rng_state)
        dist.all_gather(cuda_rng_states_all, cuda_rng_state)
    else:
        random_rng_states_all = [random_rng_state]
        cuda_rng_states_all = [cuda_rng_state]

    random_rng_states_all = torch.stack(random_rng_states_all).cpu()
    cuda_rng_states_all = torch.stack(cuda_rng_states_all).cpu()

    if local_rank == 0:
        checkpoint = {'epoch': epoch,
                      'cuda_rng_state_all': cuda_rng_states_all,
                      'random_rng_states_all': random_rng_states_all,
                      'config': config,
                      'state_dict': model.state_dict(),
                      'optimizer': optimizer.state_dict()}
        if amp_run:
            checkpoint['amp'] = amp.state_dict()

```



```python
def load_checkpoint(model, optimizer, epoch, config, amp_run, filepath, local_rank):

    checkpoint = torch.load(filepath, map_location='cpu')

    epoch[0] = checkpoint['epoch']+1
    device_id = local_rank % torch.cuda.device_count()
    torch.cuda.set_rng_state(checkpoint['cuda_rng_state_all'][device_id])
    if 'random_rng_states_all' in checkpoint:
        torch.random.set_rng_state(checkpoint['random_rng_states_all'][device_id])
    elif 'random_rng_state' in checkpoint:
        torch.random.set_rng_state(checkpoint['random_rng_state'])
    else:
        raise Exception("Model checkpoint must have either 'random_rng_state' or 'random_rng_states_all' key.")
    config = checkpoint['config']
    model.load_state_dict(checkpoint['state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer'])

    if amp_run:
        amp.load_state_dict(checkpoint['amp'])

```

TORCH.CUDA.AMP

TORCH NATIVE AMP

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

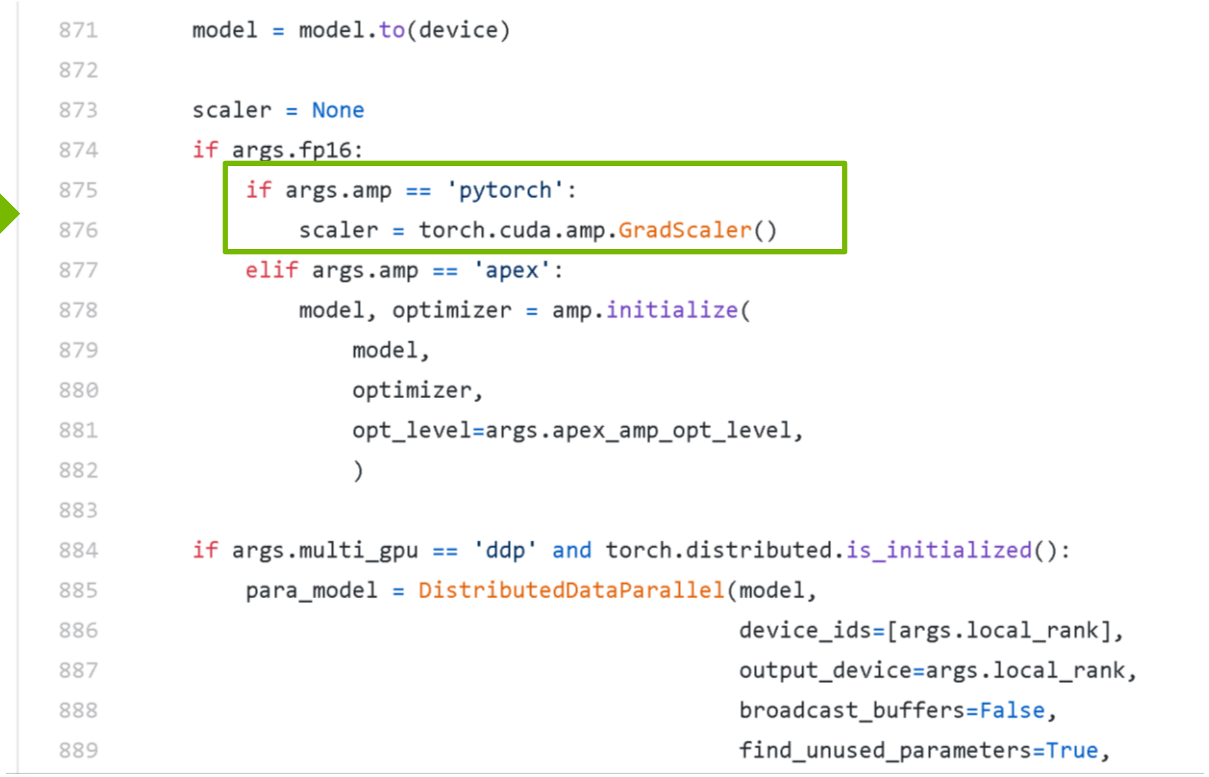
APEX.AMP Upstream to pytorch (Oct. 2020)

torch.cuda.amp.autocast(*enabled=True*)

torch.cuda.amp.GradScaler(*init_scale=65536.0, growth_factor=2.0, backoff_factor=0.5, growth_interval=2000, enabled=True*)

TORCH.CUDA.AMP

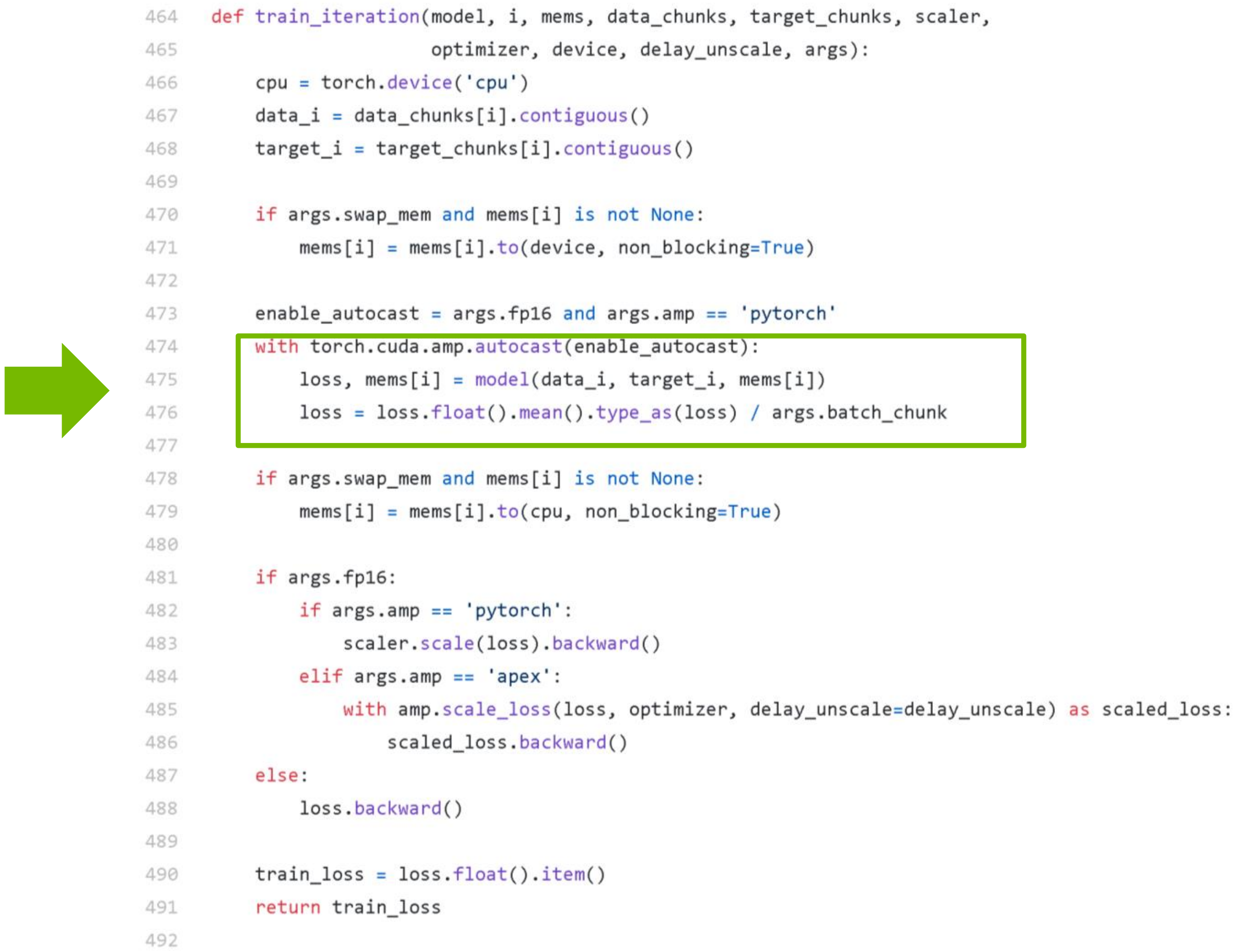
Step1. enable scaler



```python
model = model.to(device)

scaler = None
if args.fp16:
    if args.amp == 'pytorch':
        scaler = torch.cuda.amp.GradScaler()
    elif args.amp == 'apex':
        model, optimizer = amp.initialize(
            model,
            optimizer,
            opt_level=args.apex_amp_opt_level,
            )

if args.multi_gpu == 'ddp' and torch.distributed.is_initialized():
    para_model = DistributedDataParallel(model,
                                         device_ids=[args.local_rank],
                                         output_device=args.local_rank,
                                         broadcast_buffers=False,
                                         find_unused_parameters=True,
```

TORCH.CUDA.AMP

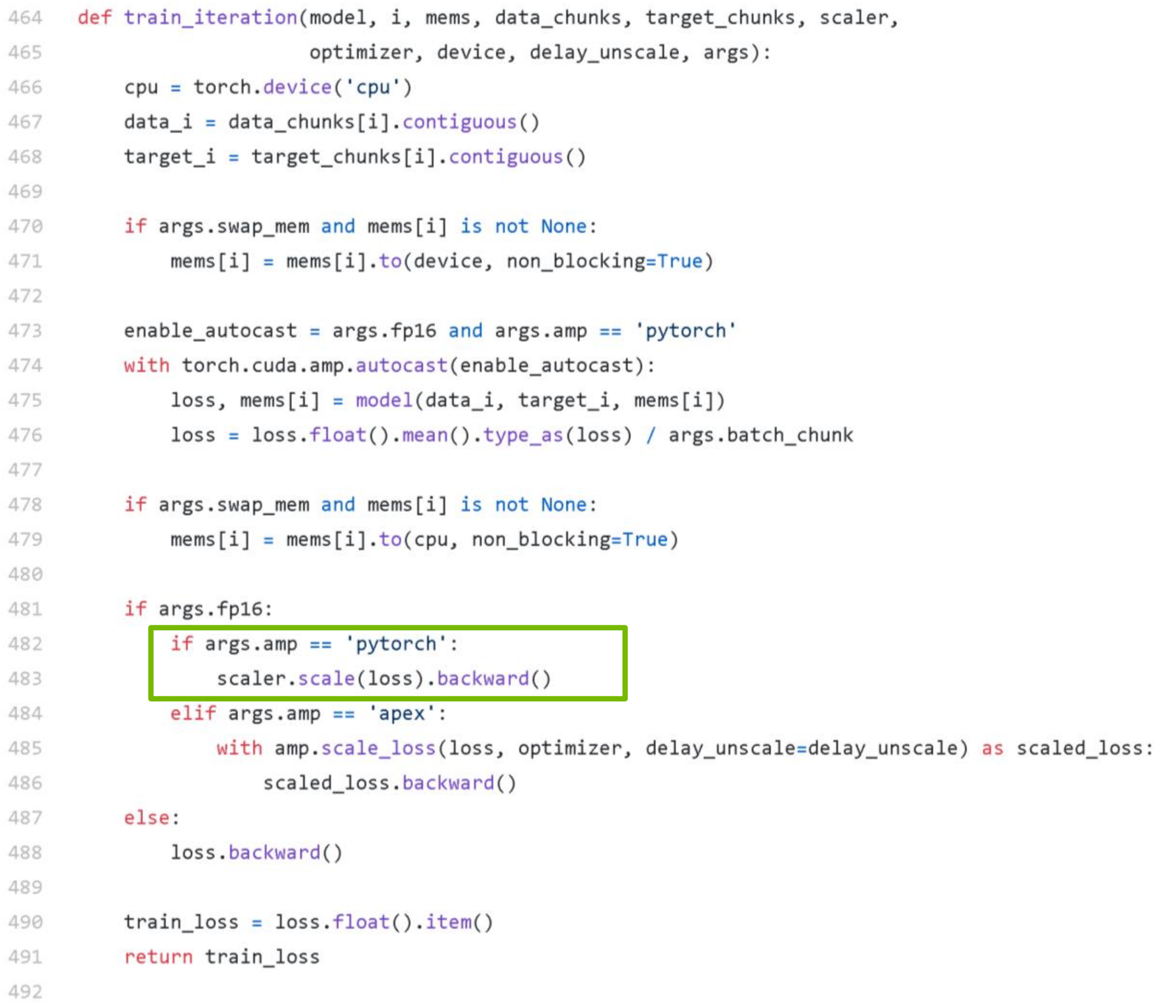
Step2. enable autocast



```python
def train_iteration(model, i, mems, data_chunks, target_chunks, scaler,
                    optimizer, device, delay_unscale, args):
    cpu = torch.device('cpu')
    data_i = data_chunks[i].contiguous()
    target_i = target_chunks[i].contiguous()

    if args.swap_mem and mems[i] is not None:
        mems[i] = mems[i].to(device, non_blocking=True)

    enable_autocast = args.fp16 and args.amp == 'pytorch'
    with torch.cuda.amp.autocast(enable_autocast):
        loss, mems[i] = model(data_i, target_i, mems[i])
        loss = loss.float().mean().type_as(loss) / args.batch_chunk

    if args.swap_mem and mems[i] is not None:
        mems[i] = mems[i].to(cpu, non_blocking=True)

    if args.fp16:
        if args.amp == 'pytorch':
            scaler.scale(loss).backward()
        elif args.amp == 'apex':
            with amp.scale_loss(loss, optimizer, delay_unscale=delay_unscale) as scaled_loss:
                scaled_loss.backward()
    else:
        loss.backward()

    train_loss = loss.float().item()
    return train_loss

```

TORCH.CUDA.AMP

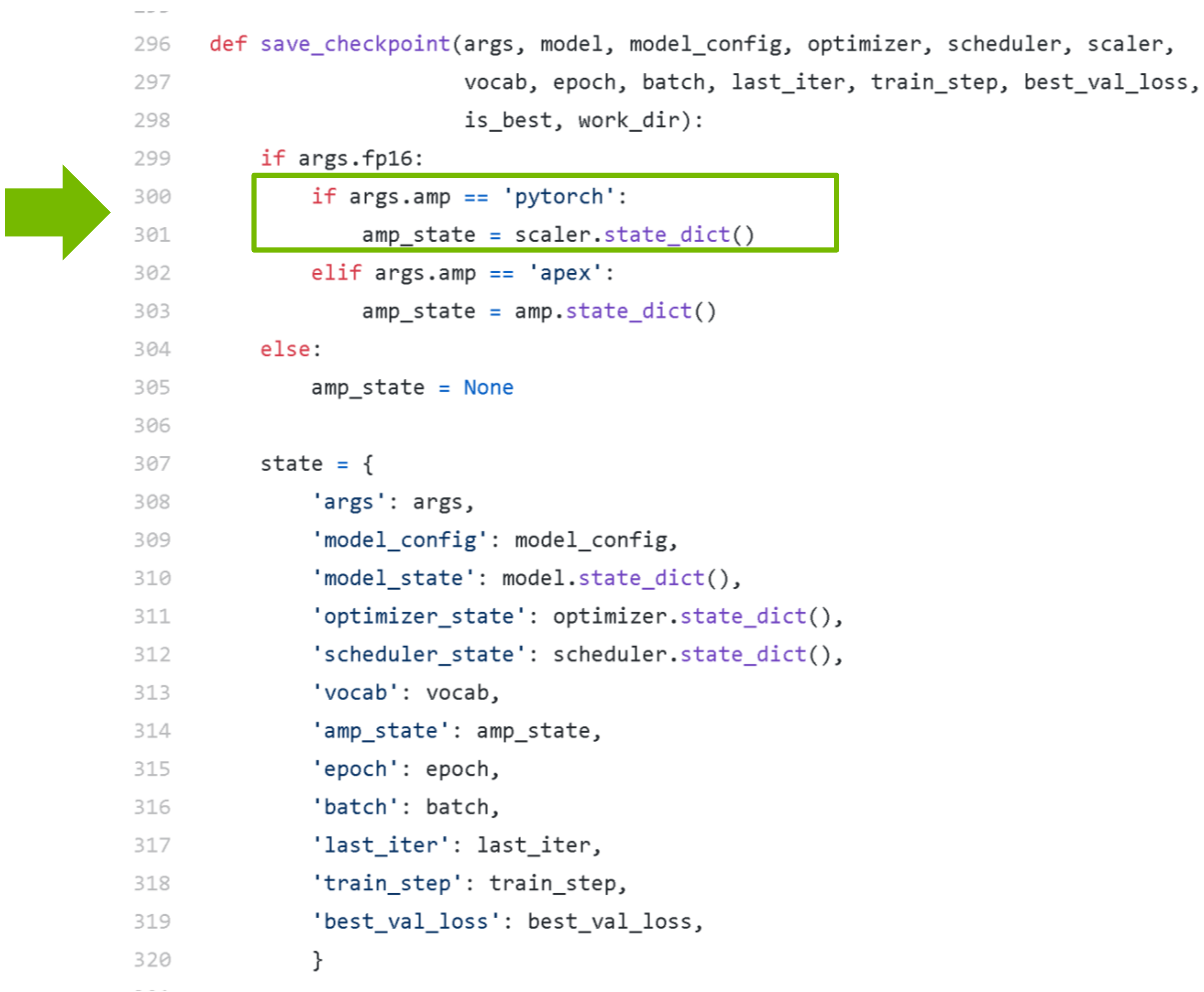
Step3. use scaler



```python
def train_iteration(model, i, mems, data_chunks, target_chunks, scaler,
                    optimizer, device, delay_unscale, args):
    cpu = torch.device('cpu')
    data_i = data_chunks[i].contiguous()
    target_i = target_chunks[i].contiguous()

    if args.swap_mem and mems[i] is not None:
        mems[i] = mems[i].to(device, non_blocking=True)

    enable_autocast = args.fp16 and args.amp == 'pytorch'
    with torch.cuda.amp.autocast(enable_autocast):
        loss, mems[i] = model(data_i, target_i, mems[i])
        loss = loss.float().mean().type_as(loss) / args.batch_chunk

    if args.swap_mem and mems[i] is not None:
        mems[i] = mems[i].to(cpu, non_blocking=True)

    if args.fp16:
        if args.amp == 'pytorch':
            scaler.scale(loss).backward()
        elif args.amp == 'apex':
            with amp.scale_loss(loss, optimizer, delay_unscale=delay_unscale) as scaled_loss:
                scaled_loss.backward()
    else:
        loss.backward()

    train_loss = loss.float().item()
    return train_loss
```

ALSO NEED

scaler.state_dict()



```python
def save_checkpoint(args, model, model_config, optimizer, scheduler, scaler,
                    vocab, epoch, batch, last_iter, train_step, best_val_loss,
                    is_best, work_dir):
    if args.fp16:
        if args.amp == 'pytorch':
            amp_state = scaler.state_dict()
        elif args.amp == 'apex':
            amp_state = amp.state_dict()
    else:
        amp_state = None

    state = {
        'args': args,
        'model_config': model_config,
        'model_state': model.state_dict(),
        'optimizer_state': optimizer.state_dict(),
        'scheduler_state': scheduler.state_dict(),
        'vocab': vocab,
        'amp_state': amp_state,
        'epoch': epoch,
        'batch': batch,
        'last_iter': last_iter,
        'train_step': train_step,
        'best_val_loss': best_val_loss,
        }
```

ADDITIONAL TIPS

Do not check loss.item() for every iteration

NVIDIA.