본 영상 교재는
2025년도 과학기술정보통신부 및 정보통신기획평가원의
SW중심대학 사업의 지원을 받아 제작되었습니다.

한국항공대학교
Korea Aerospace University

KAU
AI융합대학

SW중심대학

PyTorch 코딩

Big Data와 DDP

Big Data Training

01

Big Data Training

- Big data training
  - Training loss가 충분히 낮은 ANN이라도, 다양한 input이 제공되는 실제 상황에서도 잘 작동할 거라는 보장은 없음
  - 이를 해결하기 위해 ANN의 generalization 성능을 높여야 함
    - Generalization 성능을 높이기 위해선 충분히 다양한 데이터를 training하는 것이 좋음
  - 이 때문에 현대의 많은 deep learning 기술은 big data를 기본으로 함

- Deep learning을 위한 big data
  - 단순히 양이 많은 것이 아니라 충분히 다양한 데이터를 포함해야 함
  - 이를 위해 실제 환경 및 시뮬레이션 환경에서 충분히 다양한 데이터를 수집
  - 매우 많은 양의 데이터를 training 하는 데에는 굉장히 긴 시간이 필요함

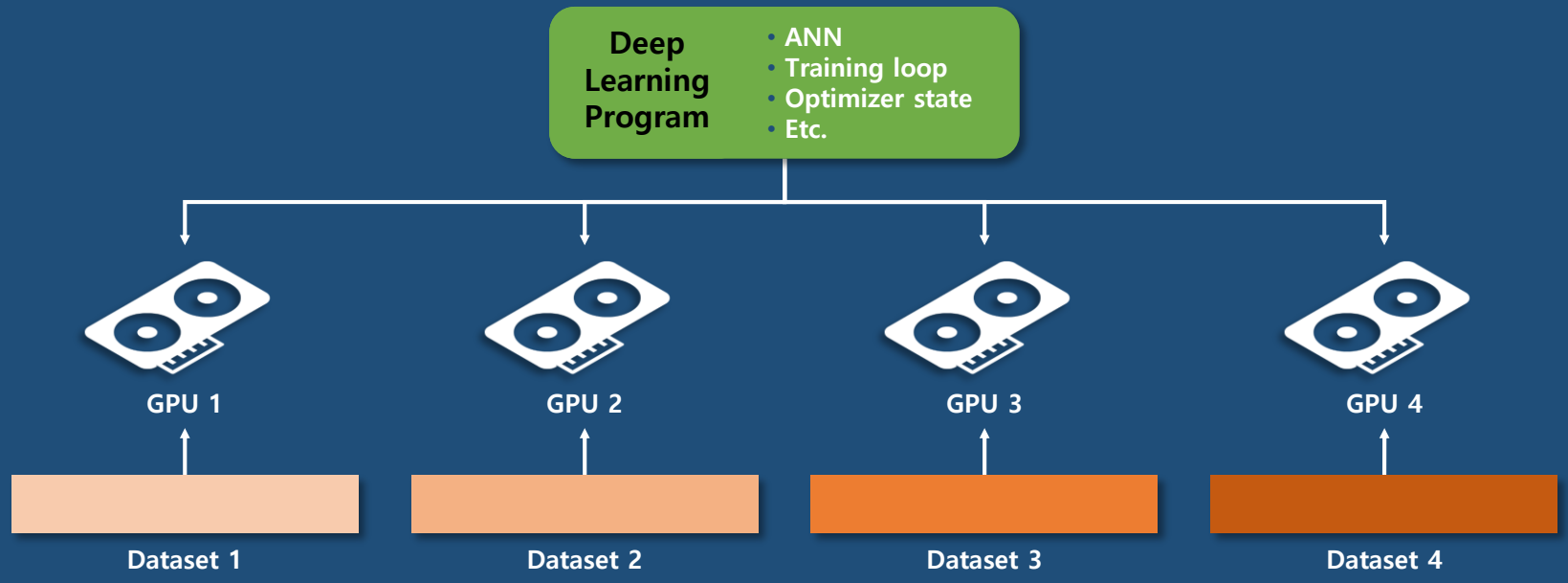
- Big data를 빠르게 training하는 방법은 없을까?

02

Distributed Data Parallel

Distributed Data Parallel

- Distributed Data Parallel (DDP)
  - Single-program / multi-data (SPMD) 방식의 PyTorch ANN training 라이브러리
  - 동일한 deep learning 프로그램(single-program)을, 여러 프로세스가 자신의 GPU를 활용하여 병렬로 동시에 실행
  - 서로 다른 dataset을 통해(multi-data) 각 프로세스가 training 작업을 수행

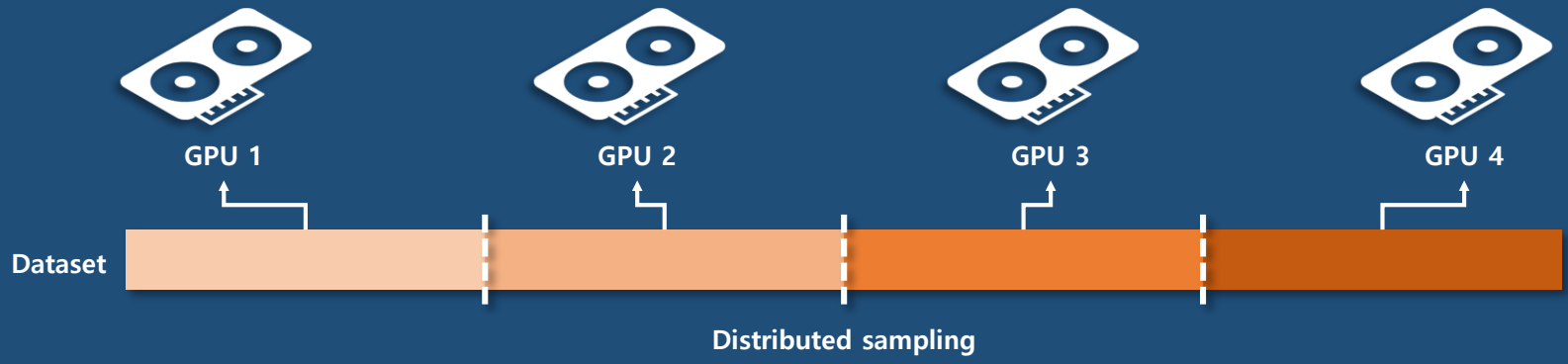


Distributed Data Parallel

- DDP 환경을 이해하기 위한 개념
  - Node : GPU를 장착한 PC 또는 서버
  - 프로세스 : Node의 각 GPU가 실행하는 프로그램
  - 프로세스 그룹 : 프로세스들이 서로 통신할 수 있게 하는 그룹
  - World size : 프로세스 그룹에 속한 프로세스의 총 개수
  - Local world size : 한 node 안에서, 프로세스에 그룹에 속한 프로세스의 개수
  - Global rank : 프로세스 그룹 전체에서 프로세스들을 식별하기 위한 정수 ID, 0부터 world size – 1까지 부여됨
  - Local rank
    - 한 node 안에서, 프로세스 그룹에 속한 프로세스들을 식별하기 위한 정수 ID
    - 각 node마다 0부터 local world size – 1까지 부여됨
    - to()를 통해 활용하려는 GPU를 지정할 때 활용할 수 있음

Distributed Data Parallel

- DDP 환경을 이해하기 위한 개념
  - Distributed sampler



- 하나의 dataset을 분산 환경에서 multi-data로 제공하기 위한 sampler
- 하나의 dataset을 프로세스 개수만큼 자름
- 잘라낸 각 dataset에서 mini-batch를 만들 수 있도록 data loader를 도와 줌
- 각 프로세스가 dataset 전체를 볼 수 있도록, 매 epoch마다 sample을 새로 섞어 dataset을 자를 수 있음

Distributed Data Parallel

- DDP 환경에서의 training 과정
  1. PyTorch의 Torchrun으로 여러 개의 프로세스를 생성
  2. 프로세스 그룹 초기화하고, 모든 프로세스가 그룹에 접속할 때까지 기다림
  3. 각 프로세스의 data loader sampling 방식을 distributed sampling 방식으로 지정
  4. 각 프로세스에서 ANN 인스턴스를 초기화한 후 DDP ANN으로 변환, 각 프로세스에 동일한 초기 파라미터를 전달

  5. 각 프로세스가 서로 다른 mini-batch를 가져와 training 진행
     - 각 프로세스는 서로 다른 mini-batch를 가지지만, 모든 프로세스의 gradient를 동기화함
       - Reduce 연산을 통해 각 프로세스의 평균 gradient를 계산해, 모든 프로세스가 공유하게 함
       - 이는 mini-batch의 크기가 커진 것과 같은 효과를 가짐
     - 동일한 초기화 작업과 동일한 gradient가 적용되어, 모든 프로세스가 동일한 상태(ANN, optimizer, scheduler 등)가 됨

  6. Training 완료 후 하나의 프로세스가 자신의 ANN parameter를 파일로 저장
     - 모든 프로세스가 동일한 ANN parameter를 가지므로 상관 없음
  7. 프로세스 그룹 제거

Distributed Data Parallel

- DDP 실습
  - Cell 1 : PyTorch DDP 라이브러리 불러오기

```
> import random
> import os                                  # PyTorch의 Torchrun이 설정한 환경 변수들을 활용하기 위한 라이브러리

> import torch
> from torch import nn
> from torch.utils.data import DataLoader
> from torch.amp import autocast, GradScaler

> import torch.distributed as dist           # Pytorch 분산 환경 라이브러리
> from torch.utils.data.distributed import DistributedSampler   # Distributed sampler 클래스
> from torch.nn.parallel import DistributedDataParallel as DDP  # PyTorch DDP 클래스

> import torchvision
> from torchvision.transforms import ToTensor
```

Distributed Data Parallel

- DDP 실습
  - Cell 3 : Data loader에 distributed sampler 적용하기

```
> def get_data_loader(distributed, dataset, mini_batch_size, shuffle):
>     if distributed:                          # 분산 환경일 경우 아래 수행
>         data_loader = DataLoader(
>             dataset,
>             batch_size = mini_batch_size,
>             pin_memory = True,
>             # shuffle = False,               # Distributed sampler를 활용하기 위해선 data loader의 shuffle을 False로 설정해야 함, False가 기본값이므로 생략 가능
>             sampler = DistributedSampler( # Distributed sampler 인스턴스를 생성하여 data loader에 연결
>                 dataset,
>                 shuffle = shuffle            # True로 설정하면 전체 dataset의 sampling 순서를 섞을 수 있음, True가 기본값
>             )
>         )
>     else:
>         data_loader = DataLoader(dataset, batch_size = mini_batch_size, pin_memory = True, shuffle = shuffle)

>     return data_loader
```

Distributed Data Parallel

- DDP 실습
  - Cell 4 : 각 프로세스의 mini-batch 크기 확인하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     @autocast(device_type = 'cuda')
>     def forward(self, x):
>         # print(f'Batch size for one GPU: {x.shape[0]}₩n', end = '') # 확인용 임시 코드, 각 프로세스에 할당되는 mini-batch의 크기를 확인할 수 있음
>
>         x1 = self.cnn_stack1(x)
>         x1 = self.linear_stack1(x1)
>
>         x2 = self.cnn_stack2(x)
>         x2 = self.linear_stack2(x2)
>
>         x = x1 + x2
>         y = self.softmax_stack(x)
>
>         return y
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-1 : Checkpoint 저장 및 불러오기 작업 변경하기

```
> def save_checkpoint(ddp_model, optimizer, scheduler):
>     checkpoint = {
>         'model_state'     : ddp_model.module.state_dict(),          # DDP ANN은 .module을 통해 접근
>         'optimizer_state' : optimizer.state_dict(),
>         'scheduler_state' : scheduler.state_dict()
>     }

>     torch.save(checkpoint, 'checkpoint.pth')

> def load_checkpoint(ddp_model, optimizer, scheduler):
>     checkpoint = torch.load('checkpoint.pth', weights_only = False)

>     ddp_model.module.load_state_dict(checkpoint['model_state'])      # DDP ANN은 .module을 통해 접근
>     optimizer.load_state_dict(checkpoint['optimizer_state'])
>     scheduler.load_state_dict(checkpoint['scheduler_state'])
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-2 : Training epoch 수정하기

```
def train(data_loader, ddp_model, optimizer, accumulation_number = 1):
    local_rank = int(os.environ['LOCAL_RANK'])        # os.environ을 통해 Torchrun이 부여한 local rank 확인
    distributed_loss = torch.zeros(2).to(local_rank)    # 모든 프로세스의 loss를 평균내기 위해 PyTorch tensor로 선언, 이를 local_rank 번째 GPU에 저장

    ddp_model.train()
    scaler = GradScaler()
    for mini_batch_index, (x, t) in enumerate(data_loader):
        x = x.to(local_rank) # Input tensor x를 local_rank 번째 GPU에 저장

        y = ddp_model(x)
        t = t.to(y.device)
        ...

            torch.nn.utils.clip_grad_norm_(ddp_model.parameters(), 1e-1)
            ...

        mini_batch_size = x.shape[0]
        distributed_loss[0] += loss.item() * mini_batch_size        # 자기 프로세스가 처리한 mini-batch의 총 loss를 누적
        distributed_loss[1] += mini_batch_size                      # 자기 프로세스가 처리한 mini-batch의 크기를 누적
    dist.all_reduce(distributed_loss, op = dist.ReduceOp.SUM)    # 모든 프로세스의 총 loss 및 mini-batch 크기를 reduce 연산으로 합산

    average_loss = distributed_loss[0] / distributed_loss[1]        # Dataset 전체의 평균 loss

    return average_loss
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-3 : Training loop에서 DDP ANN 생성하기

```
> def training_loop(dataset, mini_batch_size, max_epoch, checkpoint_interval, accumulation_number = 1):
>     world_size      = int(os.environ['WORLD_SIZE'])     # os.environ을 통해 world size 확인
>     global_rank     = int(os.environ['RANK'])           # os.environ을 통해 Torchrun이 부여한 global rank 확인
>     local_rank      = int(os.environ['LOCAL_RANK'])

>     if local_rank == 0: # Global rank 또는 local rank를 통해 특정 프로세스만 아래 작업을 수행하게 할 수 있음
>         initialize_cuda_performace_record()

>     training_data_loader = get_data_loader(distributed = True, dataset = dataset, mini_batch_size = mini_batch_size, shuffle = True)
>
>     model = NeuralNetwork().to(local_rank)     # GPU ID는 global rank가 아닌 local rank를 활용해야 함
>                                                # PyTorch 2.4의 분산 환경은 compile 기능을 지원하지 않기 때문에, 관련 코드 삭제
>     ddp_model = DDP(model)                     # 일반 ANN을 DDP ANN으로 변환, 분산 프로세스 그룹 생성 후에 수행 가능

>     optimizer = torch.optim.Adam(
>         ddp_model.parameters(),     # DDP ANN의 parameter와 연결
>         lr = 1e-2,                  # DDP 환경에서는 learning rate를 높여도 training이 덜 불안정해짐
>         betas = (0.9, 0.999),
>         weight_decay = 1e-4)

>     scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size = 1, gamma = 0.5)
>     ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-3 : Training loop에서 DDP ANN 생성하기
    - DDP 환경에서의 learning rate
      - PyTorch는 mini-batch 내 sample들의 평균 loss를 통해 back-propagation을 수행
      - DDP 환경에서는 모든 프로세스가 구한 loss의 평균으로 back-propagation을 수행
      - 따라서 프로세스가 많을 수록 mini-batch 크기가 늘어난 것과 같음

      - 이 때문에 mini-batch 내 data correlation이 줄어 learning rate를 크게 설정해도 안전함
      - 일반적으로 DDP 환경에서는 learning rate와 최대 epoch 횟수를 늘려도 parameter 진동이 쉽게 발생하지 않음

Distributed Data Parallel

- DDP 실습
  - Cell 5-4 : Training loop에서 checkpoint 불러오기

```
> def training_loop(dataset, mini_batch_size, max_epoch, checkpoint_interval, accumulation_number = 1):
>     ...

>     current_epoch = 0
>     if os.path.exists('checkpoint.pth'):
>         load_checkpoint(ddp_model, optimizer, scheduler)
>         current_epoch = scheduler.last_epoch

>         if local_rank == 0:
>             print(f'Resuming training from checkpoint at epoch {current_epoch + 1}₩n', end = '')
>         dist.barrier() # Barrier 배치, 모든 프로세스가 이 지점에 도달할 때까지 기다림

>     ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-4 : Training loop에서 checkpoint 불러오기
    - DDP 환경에서 checkpoint 불러오기
      - DDP 환경에서는 모든 프로세스가 동일한 상태(ANN, optimizer, scheduler 등)를 가져야 함
      - 따라서 모든 프로세스가 동일한 checkpoint를 불러와야 함
      - 이를 위해 NAS와 같은 공용 서버를 통해 checkpoint 파일을 공유하도록 함

      - 본 예시에서는 각 node에 동일한 checkpoint 파일을 저장

Distributed Data Parallel

- DDP 실습
  - Cell 5-4 : Training loop에서 checkpoint 불러오기
    - torch.distributed.barrier()
      - Python 병렬 프로그래밍의 join()과 같은 역할로, 모든 프로세스가 해당 지점에 도달할 때까지 기다림

      - 반드시 프로세스 그룹이 생성된 후 사용해야 함
      - 모든 프로세스가 해당 barrier에 도달해야 다음 코드로 넘어감
      - 따라서 아래와 같이 조건문을 통해 특정 프로세스에만 barrier를 배치하면, 프로그램이 더 이상 진행되지 않음

```
> if local_rank == 0:  # Local rank가 0인 프로세스만 아래 수행
>     dist.barrier()     # Local rank가 0이 아닌 프로세스는 이 barrier에 도달할 수 없기 때문에, 결국엔 모든 프로세스들이 정지함
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-5 : Training loop 수정하기

```
def training_loop(dataset, mini_batch_size, max_epoch, checkpoint_interval, accumulation_number = 1):
    ...

    for t in range(current_epoch, max_epoch):
        training_data_loader.sampler.set_epoch(t) # 매 epoch마다 distributed sampler가 sampling 순서를 섞도록 함, 분산 환경에서는 반드시 호출해야 함
        print(f'Worker {global_rank + 1} / {world_size} begins Epoch {t + 1 :> 3d} / {max_epoch}₩n', end = '')
        training_loss = train(training_data_loader, ddp_model, optimizer, accumulation_number)
        scheduler.step()
        # dist.barrier() # 분산 환경에서는 gradient 동기화 작업이 자동으로 수행되므로, barrier를 직접 배치할 필요 없음

        if local_rank == 0:
            print(f'  Training average loss: {training_loss :>8f}₩n', end = '')

            if t + 1 < max_epoch:
                print('₩n', end = '')
        dist.barrier() # Training 작업에 필요하진 않지만, 출력할 문장의 가독성을 위해 barrier 배치
        ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-5 : Training loop에서 checkpoint 저장하기

```
> def training_loop(dataset, mini_batch_size, max_epoch, checkpoint_interval, accumulation_number = 1):
>     ...
>
>     for t in range(current_epoch, max_epoch):
>         ...
>
>         if (t + 1) % checkpoint_interval == 0 and (t + 1) != max_epoch:
>             if local_rank == 0: # 각 node별로 하나의 프로세스만 checkpoint를 저장
>                 save_checkpoint(ddp_model, optimizer, scheduler)
>                 print(f'Saved training checkpoint at {t + 1} epochs to "checkpoint.pth"₩n', end = '')
>         dist.barrier() # 모든 node에 checkpoint가 저장될 때까지 기다림
>
>     ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 5-5 : Training loop에서 checkpoint 저장하기
    - DDP 환경에서 checkpoint 저장하기
      - DDP 환경에서는 모든 프로세스가 동일한 상태(ANN, optimizer, scheduler 등)를 가짐
      - 따라서 하나의 프로세스만 checkpoint 저장 작업을 수행해도 됨

Distributed Data Parallel

- DDP 실습
  - Cell 5-6 : Training loop에서 최종 ANN parameter 저장하기

```
> def training_loop(dataset, mini_batch_size, max_epoch, checkpoint_interval, accumulation_number = 1):
>     ...
>
>     for t in range(current_epoch, max_epoch):
>         ...
>
>     if global_rank == 0: # Training이 끝난 최종 ANN parameter는 하나의 프로세스만 저장해도 됨
>         excution_time, peak_VRAM_usage = get_cuda_performace_record()
>         ...
>
>         torch.save(ddp_model.module.state_dict(), 'model.pth') # DDP ANN은 .module을 통해 접근
>         print('Saved PyTorch ANN parameters to model.pth₩n', end = '')
```

Distributed Data Parallel

- DDP 실습
  - Cell 7-1 : Main 함수 정의하기

```
> if __name__ == '__main__':                                  # 분산 환경에서는 반드시 main 함수를 지정해줘야 함
>     number_of_GPU      = torch.cuda.device_count()          # 자기 node에 설치된 GPU 개수
>     world_size         = int(os.environ['WORLD_SIZE'])
>     local_world_size   = int(os.environ['LOCAL_WORLD_SIZE']) # 자기 node에서 실행된 프로세스 개수, Torchrun을 실행할 때 지정해 줄 수 있음
>     global_rank        = int(os.environ['RANK'])
>     local_rank         = int(os.environ['LOCAL_RANK'])

>     if local_rank == 0:
>         print(f'PyTorch version: {torch.__version__}₩n' +
>               f'Number of GPU: {number_of_GPU}₩n' +
>               f'World size: {world_size}₩n' +
>               f'Local world size: {local_world_size}₩n',
>               end = ''
>         )

>     if local_world_size > number_of_GPU:                    # 프로세스 개수가, GPU 개수보다 많은 경우 아래를 실행
>         if local_rank == 0:
>             print(f'Need more GPUs in this node₩n' +
>                   f'  Number of GPU in this node: {number_of_GPU}₩n' +
>                   f'  This node needs: {local_world_size}₩n',
>                   end = '')
>         exit()                                              # 자기 프로세스 종료
>     ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 7-2 : Main 함수에서 training loop 수행하기

```
> if __name__ == '__main__':
>     ...
>
>     # ---- Training ---- #
>     dist.init_process_group(   # 분산 프로세스 그룹을 생성, 모든 프로세스들이 그룹에 접속할 때까지 기다림
>         backend = 'nccl'       # 여러 GPU들이 통신할 수 있도록 통신 벡엔드를 NCCL로 설정
>     )
>
>     if local_rank == 0:  # Node당 하나의 프로세스만 dataset 파일을 다운받도록 함
>         training_dataset = get_dataset(train = True, download = True)
>     dist.barrier()          # Dataset 파일을 다운받을 때까지 모든 프로세스가 기다림
>
>     if local_rank != 0:
>         training_dataset = get_dataset(train = True, download = False)
>
>     training_mini_batch_size  = 64
>     max_epoch                 = 10
>     accumulation_number       = 4
>     checkpoint_interval       = 5
>     training_loop(training_dataset, training_mini_batch_size, max_epoch, checkpoint_interval, accumulation_number)
>
>     dist.destroy_process_group()    # Training이 끝난 후 분산 프로세스 그룹 제거
>     ...
```

Distributed Data Parallel

- DDP 실습
  - Cell 7-2 : Main 함수에서 training loop 수행하기
    - DDP 환경에서의 dataset 파일
      - DDP 환경에서는 모든 프로세스가 dataset 파일에 접근할 수 있어야 함
      - Dataset이 크다면 NAS와 같은 공용 서버를 통해 dataset 파일을 공유하도록 함

      - 본 예시에서는 dataset이 작기 때문에, 모든 node에 dataset 파일을 다운받아 활용
      - 이 경우, 각 노드의 모든 프로세스가 다운로드 작업을 중복으로 수행할 필요가 없음
      - 따라서 local rank를 통해 각 node에서 하나의 프로세스만 다운로드 작업을 수행

Distributed Data Parallel

- DDP 실습
  - Cell 7-3 : Main 함수에서 test 작업 수행하기

```
> if __name__ == '__main__':
>     ...
>
>     # ---- Test and inference ---- #
>     if global_rank == 0: # Test는 분산 환경이 필요 없기 때문에 하나의 프로세스만 test 작업 수행
>         infernece_device        = 'cuda'
>         test_dataset            = get_dataset(train = False, download = True)
>         test_mini_batch_size    = 64
>         test_data_loader        = get_data_loader(distributed = False, dataset = test_dataset, mini_batch_size = test_mini_batch_size, shuffle = False)
>
>         model = NeuralNetwork().to(infernece_device)
>         # PyTorch 2.4의 분산 환경은 compile 기능을 지원하지 않기 때문에, 관련 코드 삭제
>         model.load_state_dict(torch.load('model.pth', weights_only = True))
>
>         ...
```

Distributed Data Parallel

- DDP 실습
  - Torchrun을 통한 deep learning 프로그램 실행
    - Node가 하나뿐인 경우

```
> torchrun ₩                          # PyTorch Torchrun 실행
>     --standalone ₩                  # 하나의 node만 활용하라고 지정
>     --nproc-per-node=gpu ₩          # 현재 node에서 실행할 프로세스 개수, gpu로 설정할 경우 node에 설치된 모든 GPU 활용
>     5-ddp.py                        # 실행할 파일 이름
```

    - Node가 여러 개인 경우, 각 node에서 아래 명령어 실행
    - Hosts 파일에 동원할 node의 이름과 IP를 추가해줘야 함

```
> torchrun ₩
>     --nnodes=? ₩                    # 활용할 node 개수
>     --node-rank=? ₩                 # 현재 node의 정수 ID, 각 node는 서로 다른 ID를 가져야 함
>     --nproc-per-node=gpu ₩
>     --rdzv-id=? ₩                   # 랑데부 포인트의 정수 ID, 각 node에게 같은 ID를 알려주어야 함
>     --rdzv-backend=c10d             # 랑데부 포인트 백엔드 종류, c10d로 설정
>     --rdzv-endpoint=???.???.???.??? ₩  # 랑데부 포인트 역할을 할 node의 IP 또는 host name
>     5-ddp.py
```

Have a nice day!

Fin.