Distributed ML

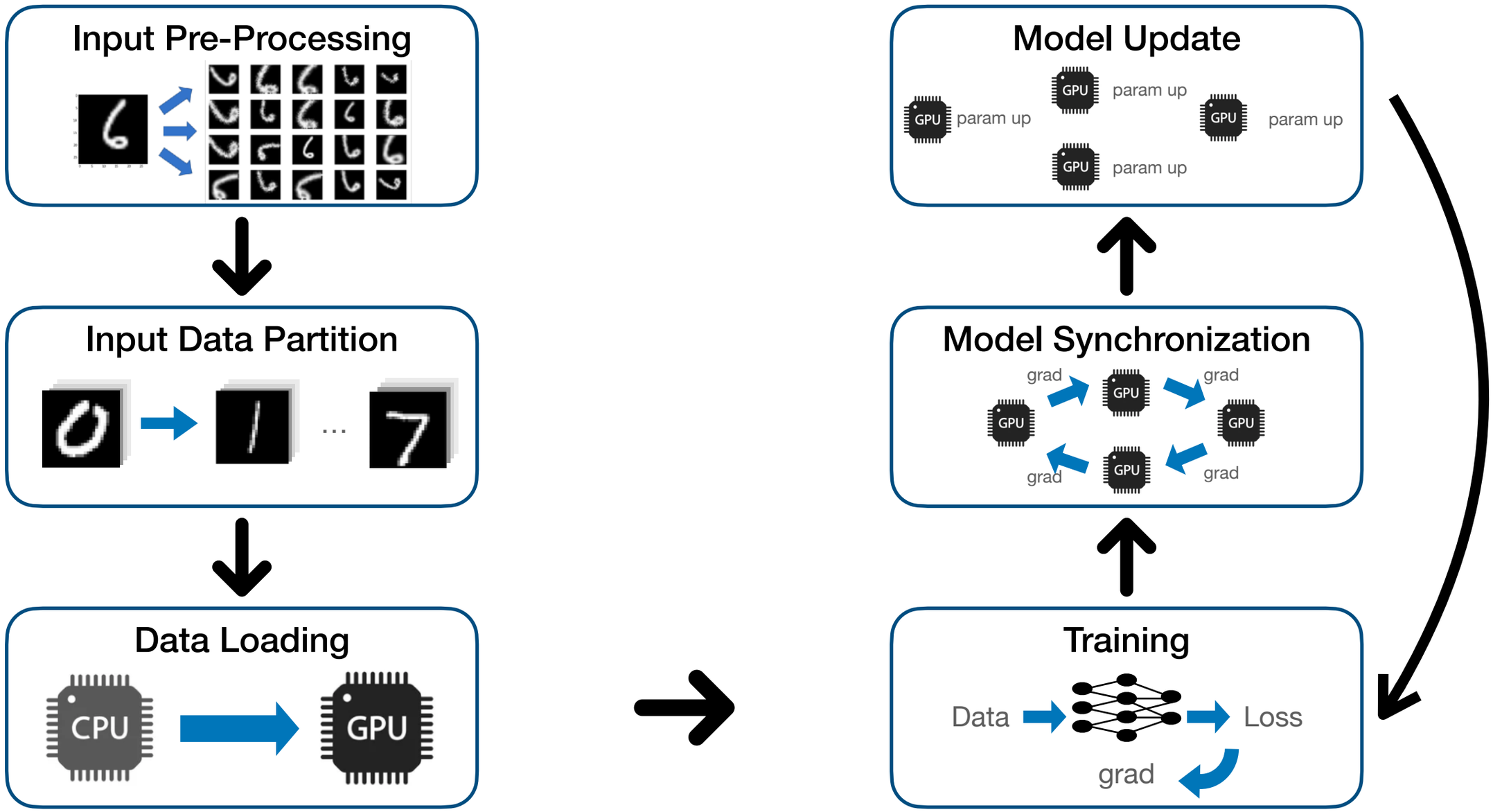
Ch 3. Building a Data Parallel Training and Serving Pipeline

Index

- The data parallel training pipeline in a nutshell
- Single-machine multi-GPUs and multi-machine multi-GPUs
- Checkpointing and fault tolerance
- Model evaluation and hyperparameter tuning
- Model serving in data parallelism

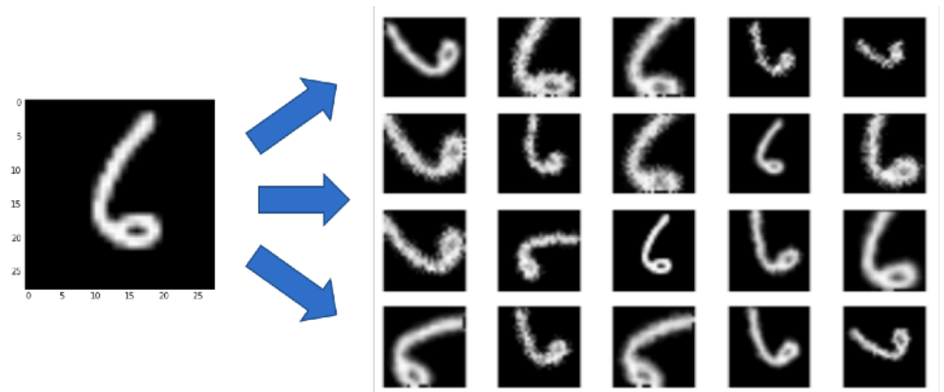
The data parallel training pipeline in a nutshell

- Pipeline overview



The data parallel training pipeline in a nutshell

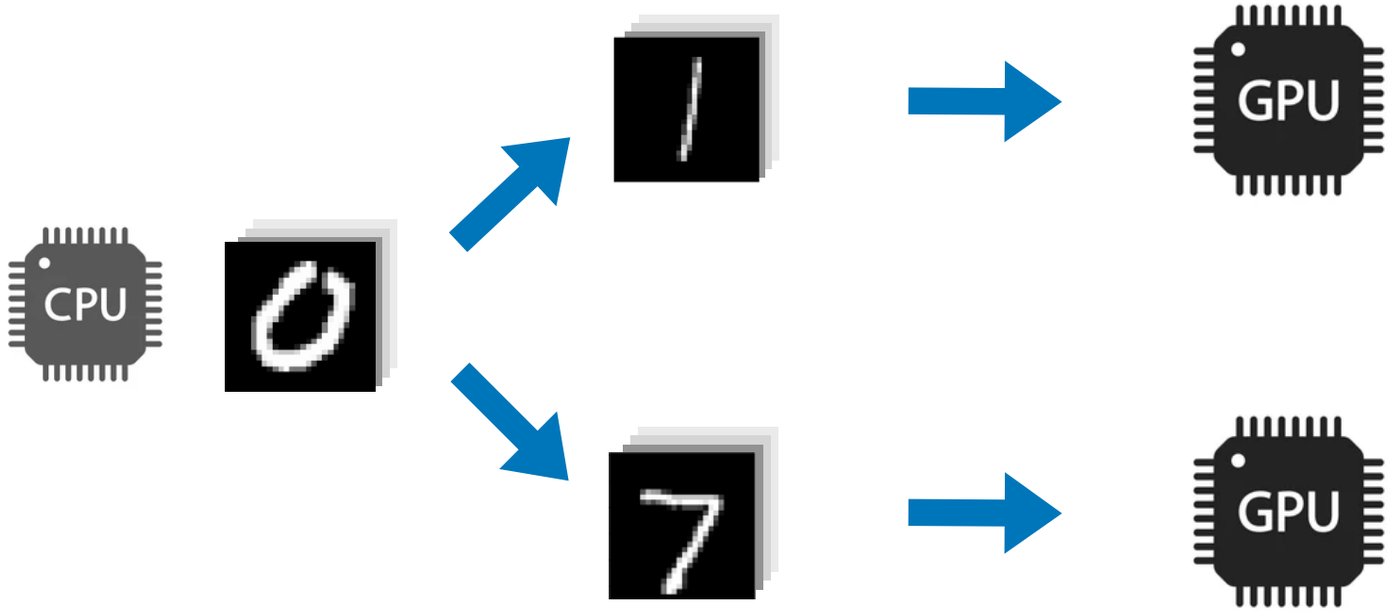
- Input pre-processing
  - Data augmentation to avoid Overfitting



```
from torchvision import transforms
# Transformations
RC = transforms.RandomCrop(32, padding=4)
RHF = transforms.RandomHorizontalFlip()
RVF = transforms.RandomVerticalFlip()
NRM = transforms.Normalize((0.1307,),(0.3081,))
TT = transforms.ToTensor()
TPIL = transforms.ToPILImage()

# Transforms object for trainset with augmentation
transform_with_aug = transforms.Compose([RC, RHF, TT, NRM])
# Transforms object for testset with NO augmentation
transform_no_aug = transforms.Compose([TT, NRM])
```

The data parallel training pipeline in a nutshell

- Input data partition & loading




```
train_dataset = datasets.MNIST(root='./data', train=True, download=True,
                               transform=transform_with_aug)
test_dataset = datasets.MNIST(root='./data', train=False, download=True,
                               transform=transform_no_aug)

# 데이터 로더
train_loader = DataLoader(train_dataset, batch_size=64, num_workers=2,
                          pin_memory=True,shuffle=True, )
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)
```

Single-machine multi-GPUs

- Training
  - nn.DataParallel 로 wrapping
  - .cuda() 함수로 GPU 메모리로 모델 복사

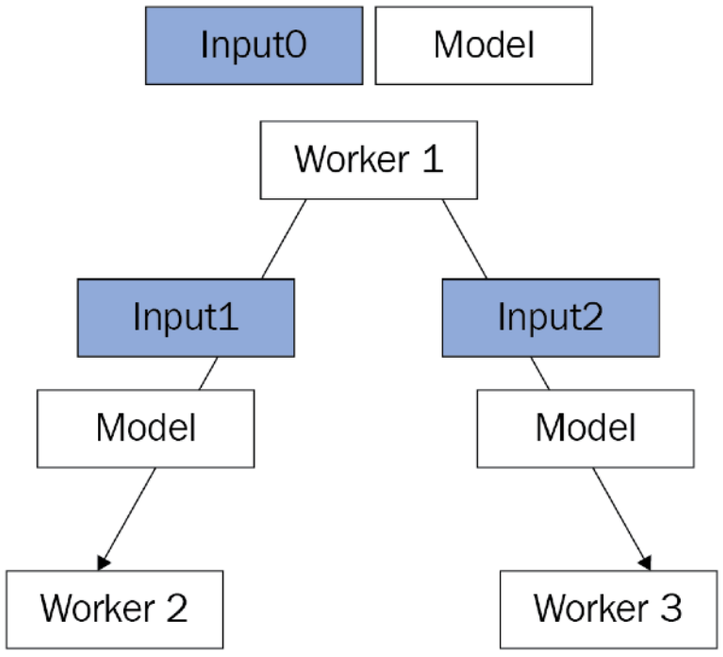
```
# 모델 초기화 및 Multi-GPU로 이동
model = NeuralNet()
model = torch.nn.DataParallel(model)
model.cuda()
```

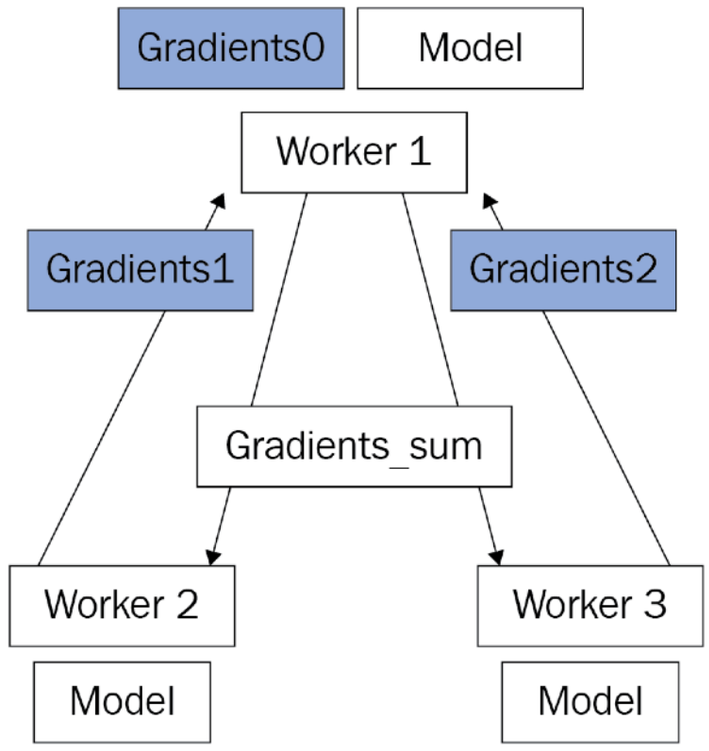
  - DataParallel 모듈에서 자동으로 병렬 학습 수행 및 Model Update

```
# 손실 함수 및 옵티마이저 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 학습 루프
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.cuda(), target.cuda()

        # Feed-Forwarding
        logit = model(data)
        loss = criterion(logit, target)

        # Back-propagation, Param update
        loss.backward()
        optimizer.step()
```

Single-machine multi-GPUs

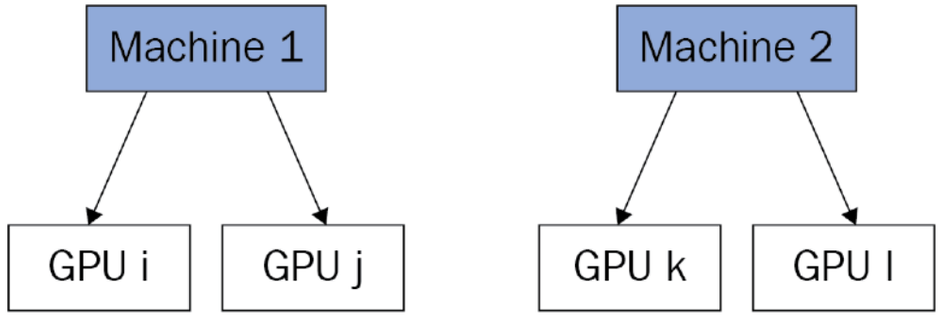
- Bottleneck of *‘nn.DataParallel()’*
  - torch의 ‘*nn.DataParallel()*’ 은 Parameter Server 방식으로 동작
  - 또한, 각 GPU에서 Multi-threading 으로 동작하기에 실제로는 순차적 동작





Multi-machine multi-GPU

- DDP (Distributed Data Parallel)



local rank:
0 1 0 1

rank:
0 1 2 3

world_size:
4

```
mp.spawn(train, nprocs=args.gpus, args=(args,), join=True)
```

```
def train(local_rank, args):

    world_size = args.machines*args.gpus
    rank = args.mid * args.gpus + local_rank
    dist.init_process_group('nccl', rank =rank, world_size = world_size,
                            timeout=datetime.timedelta(seconds=60))
    torch.cuda.set_device(local_rank)
```

Multi-machine multi-GPU

- DP VS DDP

```python
# 모델 초기화 및 Multi-GPU로 이동
model = NeuralNet()
model = torch.nn.DataParallel(model)
model.cuda()

num_epochs = 5

# 손실 함수 및 옵티마이저 정의
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 학습 루프
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.cuda(), target.cuda()

        # Feed-Forwarding
        logit = model(data)
        loss = criterion(logit, target)

        # Back-propagation, Param update
        loss.backward()
        optimizer.step()
```

```python
model = MyNet()
model = DDP(model, device_ids=[local_rank])

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 5e-4)

# 학습 루프
for epoch in range(args.epochs):
    for batch_idx, (data, target) in enumerate(local_train_loader):
        data, target = data.cuda(), target.cuda()

        # Feed-Forwarding
        logit = model(data)
        loss = criterion(logit, target)

        # Back-propagation, Param update
        loss.backward()
        optimizer.step()
```

Checkpointing and fault tolerance

- Model checkpointing

```
def checkpointing(rank, epoch, net, optimizer, loss):
    path = f"model{rank}.pt"
    torch.save({
                'epoch':epoch,
                'model_state':net.state_dict(),
                'loss': loss,
                'optim_state': optimizer.state_dict(),
                }, path)
    print(f"Checkpointing model {rank} done.")

def load_checkpoint(rank, machines):
    path = f"model{rank}.pt"
    checkpoint = torch.load(path)
    model = torch.nn.DataParallel(MyNet(), device_ids=[rank%machines])
    optimizer = torch.optim.SGD(model.parameters(), lr = 5e-4)

    epoch = checkpoint['epoch']
    loss = checkpoint['loss']
    model.load_state_dict(checkpoint['model_state'])
    optimizer.load_state_dict(checkpoint['optim_state'])
    return model, optimizer, epoch, loss
```

Model evaluation and hyperparameter tuning

- HyperParameter
  - 가중치 외에 학습에 영향을 주는 매개변수들
    (ex. Batch-size, learning rate, layer nums, etc.)
  - Grid search, Random search 와 같은 방법으로 최적의 조합을 서치
  - Katib, Optuna 와 같은 오픈소스에서 자동화 지원