



# Distributed Training 100

딥러닝 분산 학습, 일주일만 하면 컴린이만큼 한다.

Daekeun Kim

AIML Specialist Solutions Architect  
Worldwide Specialist Organization, AWS

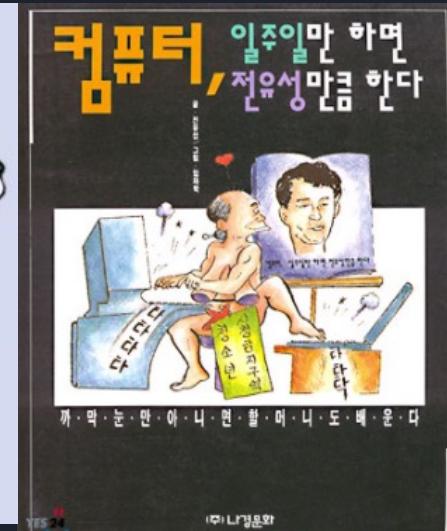
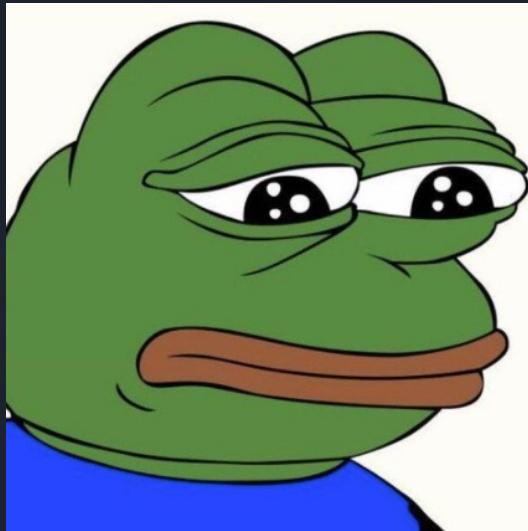
# Target audience

- 훈련 속도가 너~무 느려요 (X)
- 모델이 GPU 메모리에 다 안 들어가요 (X)
- 큰 맘 먹고 시작해 보려는데 외계어에요 ㅠㅠ (O)



# 시작하기 전: 꿈과 희망을 가집시다!

개념을 알면 절반은 먹고 들어갑니다.



Before: Today

After: One week later (Non-tech)

One day (tech)

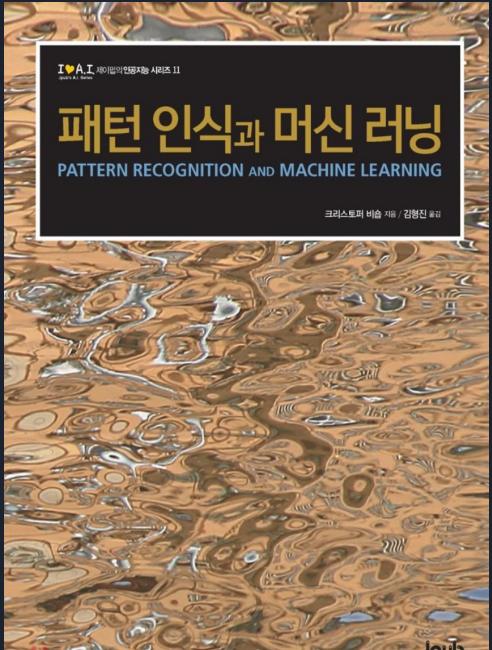
One hour (ML-related tech)

# 0. 준비 운동



# 과거 (라떼는 말이야~)

- 수학 역량 필요 (Baby Rudin, Bishop), 선증명 & 후실험



Given  $\phi = \{\phi, \mu_0, \mu_1, \Sigma\}$ . Then the likelihood function is

$$\log \prod_{i=1}^N P(X_i|y_i=0) = \log \prod_{i=1}^N P(X_i|y_i=0) P(y_i=0)$$

$$= \log \left[ \prod_{i=1}^N P(X_i|y_i=0) P(y_i=0) \right] = \sum_{i=1}^N \log P(X_i|y_i=0) + \log P(y_i=0)$$

$$= \sum_{i=1}^N [\log P(X_i|y_i=0) + \log P(y_i=0)] + \log P(y_i=1)$$

$$+ \sum_{i=1}^N [\log P(X_i|y_i=1) I[y_i=1] + \log P(y_i=1) I[y_i=1]]$$

(note that  $I[y_i=0] = \begin{cases} 1 & \text{if } y_i=0 \\ 0 & \text{if } y_i \neq 0 \end{cases}$ ,  $I[y_i=1] = \begin{cases} 0 & \text{if } y_i=0 \\ 1 & \text{if } y_i=1 \end{cases}$ )

$$= \sum_{i=1}^N \left( \log \frac{1}{(2\pi)^{1/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (x_i - \mu_0)^T \Sigma^{-1} (x_i - \mu_0) \right) I[y_i=0] + \log (1 - \phi) I[y_i=0] \right)$$

$$+ \sum_{i=1}^N \log \frac{1}{(2\pi)^{1/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (x_i - \mu_1)^T \Sigma^{-1} (x_i - \mu_1) \right) I[y_i=1] + \log \phi I[y_i=1]$$

(by assumption of the problem)

$$= \sum_{i=1}^N \left[ -\frac{1}{2} \log(2\pi) - \frac{1}{2} (\mu_0 - \mu_1)^T \Sigma^{-1} (\mu_0 - \mu_1) \right] I[y_i=0] + \log(1 - \phi) I[y_i=0]$$

$$+ \sum_{i=1}^N \left[ -\frac{1}{2} \log(2\pi) - \frac{1}{2} (\mu_0 - \mu_1)^T \Sigma^{-1} (\mu_0 - \mu_1) \right] I[y_i=1] + \log \phi I[y_i=1] \quad \dots$$

Assume  $M(\phi)$ . Then  $\phi$  can be written as follows:

$$M(\phi) = \sum_{i=1}^N \left[ -\frac{1}{2} \log(2\pi) - \frac{1}{2} (\mu_0 - \mu_1)^T \Sigma^{-1} (\mu_0 - \mu_1) \right] I[y_i=0] + \log(1 - \phi) I[y_i=0] + \sum_{i=1}^N \left[ -\frac{1}{2} \log(2\pi) - \frac{1}{2} (\mu_0 - \mu_1)^T \Sigma^{-1} (\mu_0 - \mu_1) \right] I[y_i=1] + \log \phi I[y_i=1] \quad \dots$$

First, we need to show for  $\phi$ . Taking the gradient of  $M(\phi)$  with respect to  $\phi$ , we have

$$\nabla \phi M(\phi) = \nabla \phi \left( \sum_{i=1}^N \log(1 - \phi) I[y_i=0] + \sum_{i=1}^N \log \phi I[y_i=1] \right) \quad (\text{other terms are constants})$$

$$= \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=0] + \frac{1}{\phi} \sum_{i=1}^N I[y_i=1]$$

Setting  $\nabla \phi M(\phi) = 0$ , we get

$$0 = -\frac{1}{1-\phi} \sum_{i=1}^N I[y_i=0] + \frac{1}{\phi} \sum_{i=1}^N I[y_i=1] \Rightarrow \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=0] = \frac{1}{\phi} \sum_{i=1}^N I[y_i=1]$$

$$\Rightarrow \phi \sum_{i=1}^N I[y_i=0] = \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=1] - \phi \sum_{i=1}^N I[y_i=1]$$

$$\Rightarrow \phi \sum_{i=1}^N I[y_i=0] + \phi \sum_{i=1}^N I[y_i=1] = \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=1]$$

$$\Rightarrow \phi ( \sum_{i=1}^N I[y_i=0] + \sum_{i=1}^N I[y_i=1] ) = \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=1]$$

$$\Rightarrow \phi N = \frac{1}{1-\phi} \sum_{i=1}^N I[y_i=1] \quad \dots$$

$$\Rightarrow \phi = \frac{N}{N+1} \sum_{i=1}^N I[y_i=1] \quad \dots$$

Given  $p(t|X, w, \beta)$  is  $\prod_{i=1}^N N(x_i|w^T \phi(x_i), \beta^{-1})$  (defined by 3.10)

From  $p(w) \propto N(w|m_0, S_0)$  (by 2.9), we have:

$$p(w|t) \propto \left( \prod_{i=1}^N N(x_i|w^T \phi(x_i), \beta^{-1}) \right) N(w|m_0, S_0)$$

$$= N(t | \phi(w), \beta^{-1}) N(w|m_0, S_0)$$

(where  $\phi$  is a NUM design matrix)

$$= C \cdot \exp \left( -\frac{1}{2} (t - \phi(w))^T (t - \phi(w)) \right) \exp \left( -\frac{1}{2} (w - m_0)^T S_0^{-1} (w - m_0) \right)$$

$$\propto \exp \left( -\frac{1}{2} (t - \phi(w))^T (t - \phi(w)) - \frac{1}{2} (w - m_0)^T S_0^{-1} (w - m_0) \right)$$

$$= \exp \left( -\frac{1}{2} \left[ \phi(t - \phi(w))^T (t - \phi(w)) + (w - m_0)^T S_0^{-1} (w - m_0) \right] \right)$$

$$= \exp \left( -\frac{1}{2} \left[ (\phi t^T - \phi \phi^T t) + (w^T S_0^{-1} w - w^T S_0^{-1} m_0 + m_0^T S_0^{-1} w + m_0^T S_0^{-1} m_0) \right] \right)$$

$$= \exp \left( -\frac{1}{2} \left[ w^T S_0^{-1} w - \phi^T \phi w - \phi^T S_0^{-1} w + \phi^T t + w^T S_0^{-1} m_0 - \phi^T \phi m_0 - \phi^T t + m_0^T S_0^{-1} w + m_0^T S_0^{-1} m_0 \right] \right)$$

$$= \exp \left( -\frac{1}{2} \left[ w^T S_0^{-1} w - \phi^T \phi w - \phi^T S_0^{-1} w + \phi^T t + w^T S_0^{-1} m_0 - \phi^T \phi m_0 - \phi^T t + m_0^T S_0^{-1} w + m_0^T S_0^{-1} m_0 \right] \right)$$

(Note that  $a^T b = b^T a$  if  $a$  and  $b$  are vecors)

$$= \exp \left( -\frac{1}{2} \left[ w^T (S_0^{-1} \phi^T \phi) w - (\phi^T \phi^T + m_0^T S_0^{-1} m_0) t w - w^T (S_0^{-1} \phi^T m_0) + \phi^T m_0^T S_0^{-1} w \right] \right) \quad \dots$$

$$= \exp \left( -\frac{1}{2} \left[ w^T S_0^{-1} w - (S_0^{-1} m_0)^T w - w^T (S_0^{-1} \phi^T \phi) w + \phi^T m_0^T S_0^{-1} w \right] \right)$$

(Since  $m_0 = S_0 (S_0^{-1} m_0 + \phi^T \phi t)$  by 2.20)

$$= \exp \left( -\frac{1}{2} \left[ (w^T S_0^{-1} w - 2(S_0^{-1} m_0)^T w + m_0^T S_0^{-1} m_0) \right] \right) \quad \text{for making quadratic form} + \phi^T t + m_0^T S_0^{-1} m_0 - m_0^T S_0^{-1} m_0$$

$$= \exp \left( -\frac{1}{2} \left[ (w - m_0)^T S_0^{-1} (w - m_0) + \phi^T t + m_0^T S_0^{-1} m_0 - m_0^T S_0^{-1} m_0 \right] \right)$$

$$= \exp \left( -\frac{1}{2} \left[ (w - m_0)^T S_0^{-1} (w - m_0) + \phi^T t + m_0^T S_0^{-1} m_0 - m_0^T S_0^{-1} m_0 \right] \right) \quad \text{normalization factor}$$

Therefore, the result (3.49) is verified.  $\square$

# 과거 (라떼는 말이야~)

- Given example images  $(x_1, y_1), \dots, (x_n, y_n)$  where  $y_i = 0, 1$  for negative and positive examples respectively.
- Initialize weights  $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$  for  $y_i = 0, 1$  respectively, where  $m$  and  $l$  are the number of negatives and positives respectively.
- For  $t = 1, \dots, T$ :

- Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

so that  $w_t$  is a probability distribution.

- For each feature,  $j$ , train a classifier  $h_j$  which is restricted to using a single feature. The error is evaluated with respect to  $w_t$ ,  $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$ .
- Choose the classifier,  $h_t$ , with the lowest error  $\epsilon_t$ .
- Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-\epsilon_i}$$

where  $e_i = 0$  if example  $x_i$  is classified correctly,  $e_i = 1$  otherwise, and  $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ .

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where  $\alpha_t = \log \frac{1}{\beta_t}$

- 오픈 소스가 없으므로 논문의 알고리즘과 수학 수식을 참조하여 밑바닥부터 직접 구현 및 검증
  - Integral Image
  - Haar Like features
  - Adaboost
  - Cascading 등
- Multi-object 검출은 고난의 행군

Source: Rapid Object Detection using a Boosted Cascade of Simple Features  
(<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>)

# 현재

- 평균, 표준편차만 알아도 절반 먹고 들어감
- 선실험, 후증명(1% 미만의 Researcher)
- API 호출로 AIML 쉽게 접근 가능, 수많은 오픈 소스들
- 3달 (2001년) vs. 3분 (2021년)

Object Detection 모델 로드하고~

```
net = model_zoo.get_model('ssd_512_resnet50_v1_voc', pretrained=True)
```

입력 데이터 전처리하고~

```
x, img = data.transforms.presets.ssd.load_test(im_fname, short=512)
```

추론하고~ 참 쉽죠?

```
class_ids, scores, bounding_boxes = net(x)
```

[https://gluon-cv.mxnet.io/build/examples\\_detection/](https://gluon-cv.mxnet.io/build/examples_detection/)

# 하지만!!

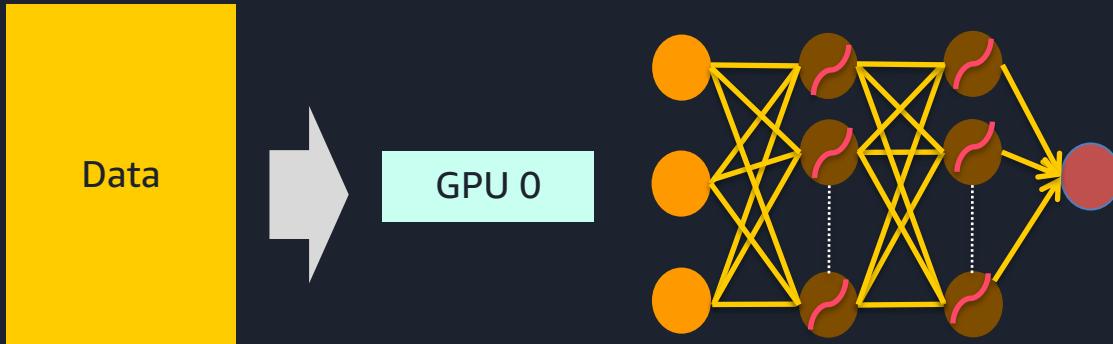
## 밑바닥부터 딥러닝 모델 훈련

- 또다른 노가다
  - 피쳐 엔지니어링: 모델링 = ??:?
  - 디버깅... 디버깅... 이하 반복
  - 모니터링... 모니터링... 이하 반복
- 마른 수건 쥐어짜기
  - GPU만 많이 구매하면 끝?
  - 하드웨어 엔지니어의 마인드
- 관리형 서비스의 적극 활용
  - AWS != AIML 전문 회사

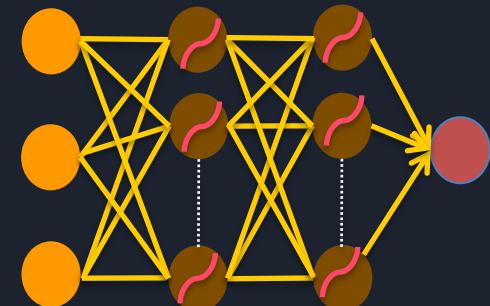
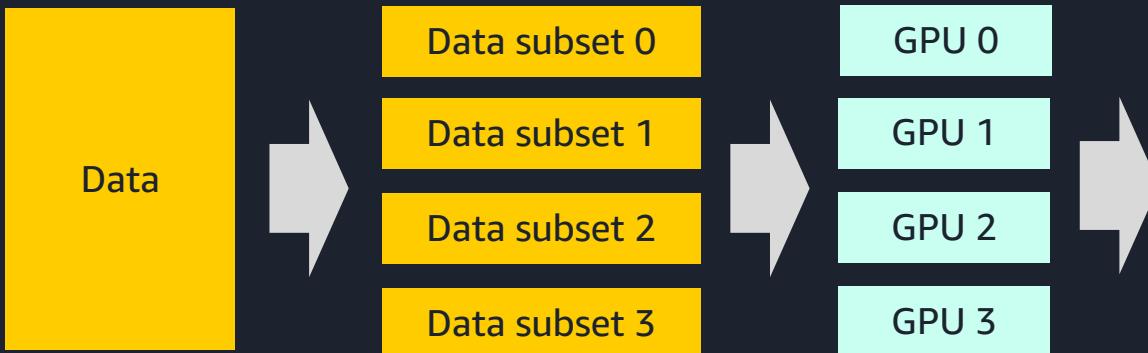
## 프로덕션에 모델 배포

- 기본 개념 체화 (반복, 또 반복)
  - 복잡한 수학/통계? No!
  - 딥러닝 & Boosted Tree가 킹왕짱? No!
- 비판적 사고 & 포용력
  - 모수의 함정 (Simpsons Paradox)
  - 포텐셜 충만한 주니어 >>> 20년차 AIML 경력자
- 관리형 서비스의 적극 활용
  - AWS != AIML 전문 회사

# Topic: Single GPU and Data Parallelism



Q: 4x faster?



# Agenda

1. Single GPU
2. Multi GPUs (Single Machine)
3. Multi GPUs (Multi Machines)
4. Collective Communication
5. Hardware Communication

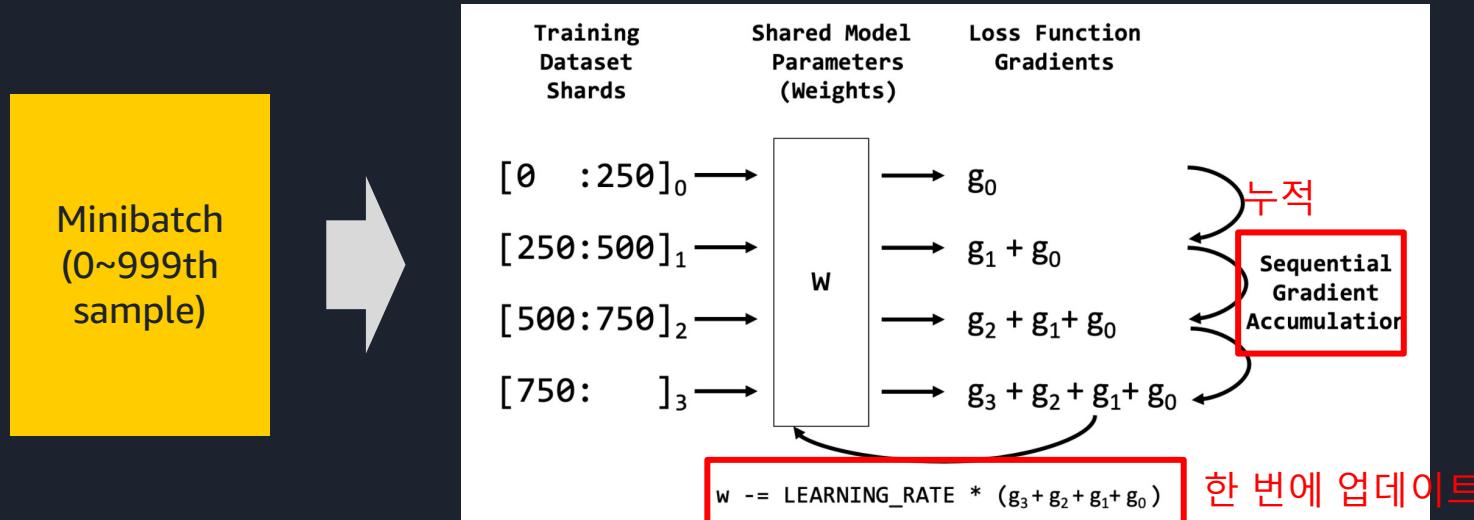
# 1. Single GPU

# 생각해 봅시다 (1)

- 금붕어 데이터 과학자
  - BERT 모델을 파인튜닝하고 싶은데 메모리가 부족하네? 속도도 느리네? GPU 구매 품의 올리자~
- (약간) 깐깐한 강아지 구매팀 담당자:
  - GPU/Memory Utilization이 계속 100%인 증빙 자료가 필요합니다.
  - Swapping, Dynamic Programming 처럼 리소스를 효율적으로 사용하는 방법들은 고려해 보셨나요?
- 닭 잡는데 소 잡는 칼? No!

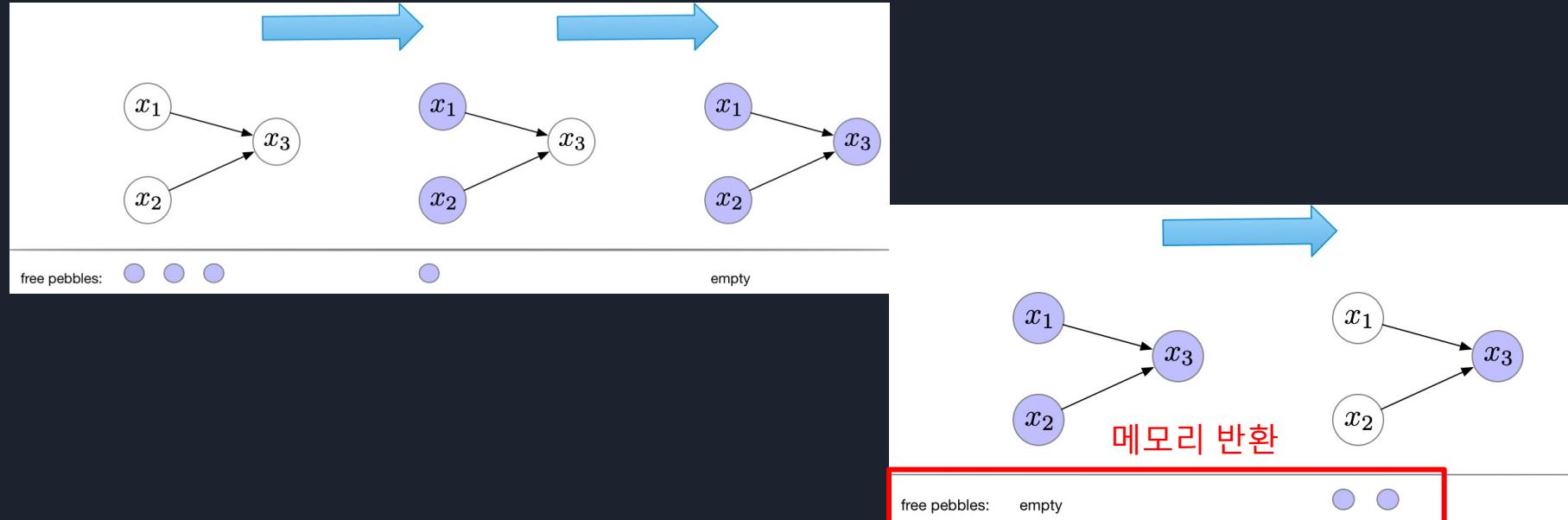
# Gradient Accumulation

- 부족한 GPU의 메모리에서 batch size를 키우기 위한 방법
- 매 step마다 파라메터를 업데이트하지 않고(optimizer.step() 호출) parameter.grad 텐서에서 여러 backward 연산의 gradient들을 모았다가 일정 step이 경과하면 파라메터를 업데이트
- loss도 합산되므로 accumulation\_steps로 나눠야 함



# Gradient Checkpoint

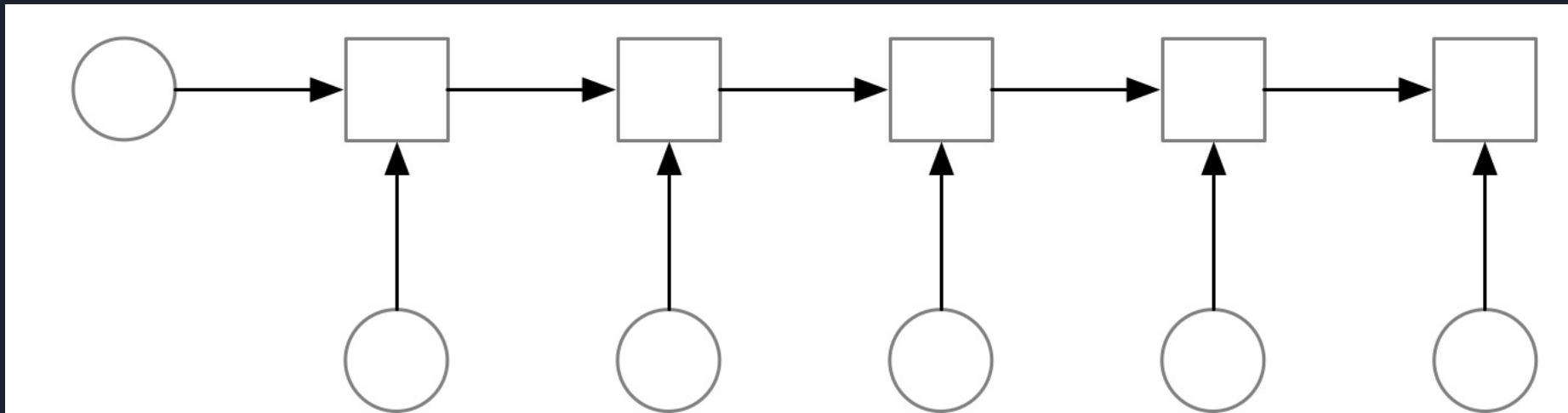
- 최초 소개: Training Deep Nets with Sublinear Memory Cost  
(<https://arxiv.org/pdf/1604.06174.pdf>)
- Gradient를 노드마다 저장하는 대신, 그래프를 분석하여 저장되지 않아도 되는 gradient를 삭제



# Gradient Checkpoint Idea (Optional)

- 필요한 노드에만 조약돌(pebbles, 메모리)을 놓고 중간 결과 계산
- 값이 더 필요하지 않으면 노드에서 조약돌을 제거하고 향후 계산에 사용
- Memory Requirement:  $O(\sqrt{n})$ 
  - $O(1)$  메모리 전략을 사용할 수도 있지만, 계산 복잡도가  $O(n^2)$ 로 증가하므로 권장하지 않음
  - $O(\sqrt{n})$  전략으로 중간 결과를 체크포인트로 지정하여 저장하면 계산 복잡도가  $O(n)$ 이기에 권장 전략으로 사용
- 속도는 약 10~20% 정도 느려지지만, Transformer 같은 대용량 모델 학습에 필요한 메모리를 크게 줄일 수 있으며, 긴 sequence의 RNN에 대해서도 효과적
- 계산 그래프 적용을 위해 Dynamic programming 및 트리 분해 기법 적용

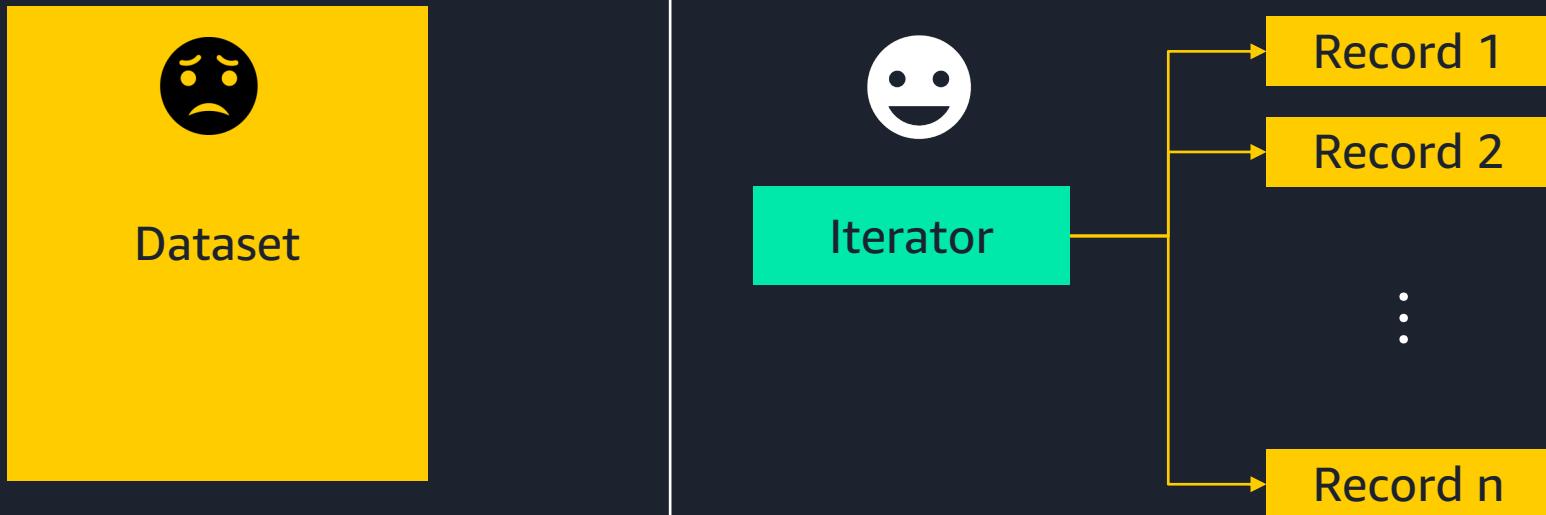
# Gradient Checkpoint Example



출처: <https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9>

# Iterable-style Dataset

- CPU 리소스 및 리소스 절약
- `__getitem__()`과 `__len__()`을 구현하는 대신 `__iter__()` 구현
- map-style dataset은 `dataset[index]`를 사용하는 반면 iterable-style dataset은 `next(iterator_dataset)`을 사용하므로, 데이터셋을 사전에 셔플링하는 것이 좋음

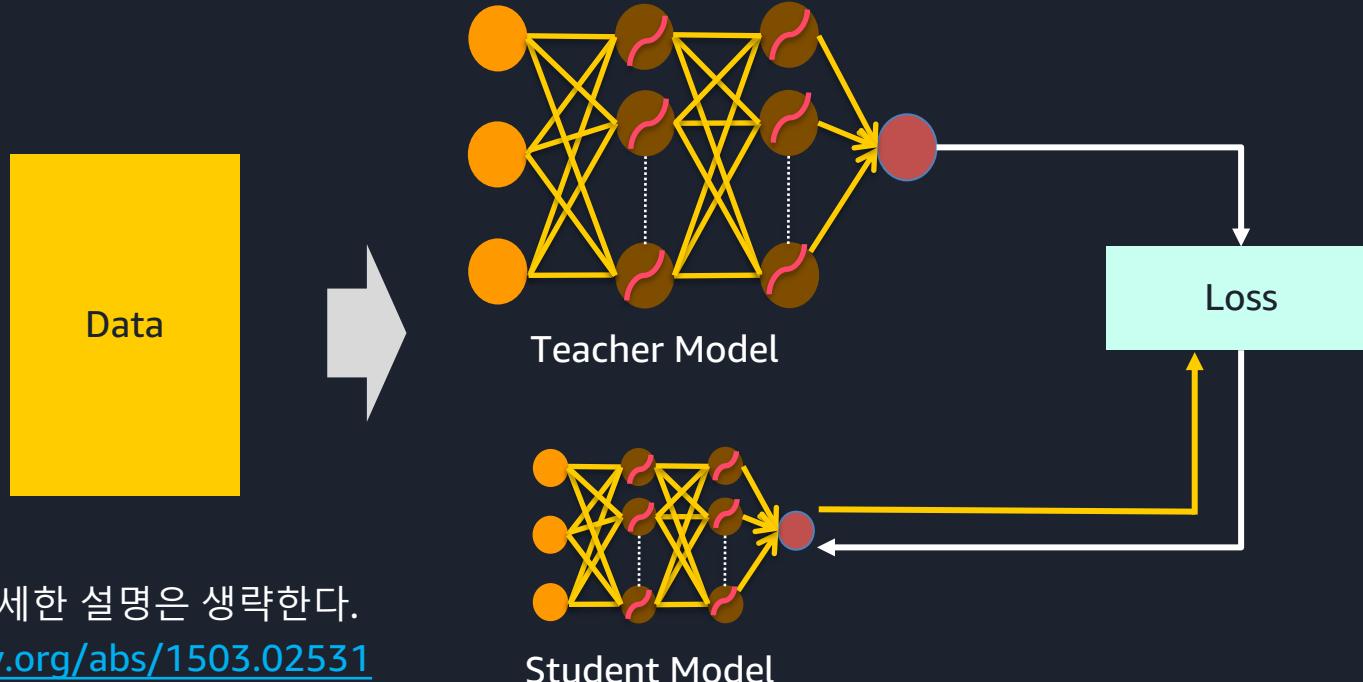


# 생각해 봅시다 (2)

- Andrew Ng 신봉하는 신입 데이터 과학자
  - 1epoch 훈련 시 데이터가 천만 건 필요한데 너무 느려요.
  - 데이터가 많을수록 좋다는데 GPU 1장으로 벅차요 ㅠㅠ
- 현실주의 강조하는 선임 데이터 과학자
  - 샘플링 후 Baseline 모델링 해보셨어요?
  - 모델 경량화 & 모델 양자화도 충분히 검토해 보셨죠?
  - Memory Pinning도 적용하셨죠?
- 닭 잡는데 소 잡는 칼? No!

# Model Distillation: 청출어람

- Teacher의 모델과 student 모델의 확률분포 차이를 최소화

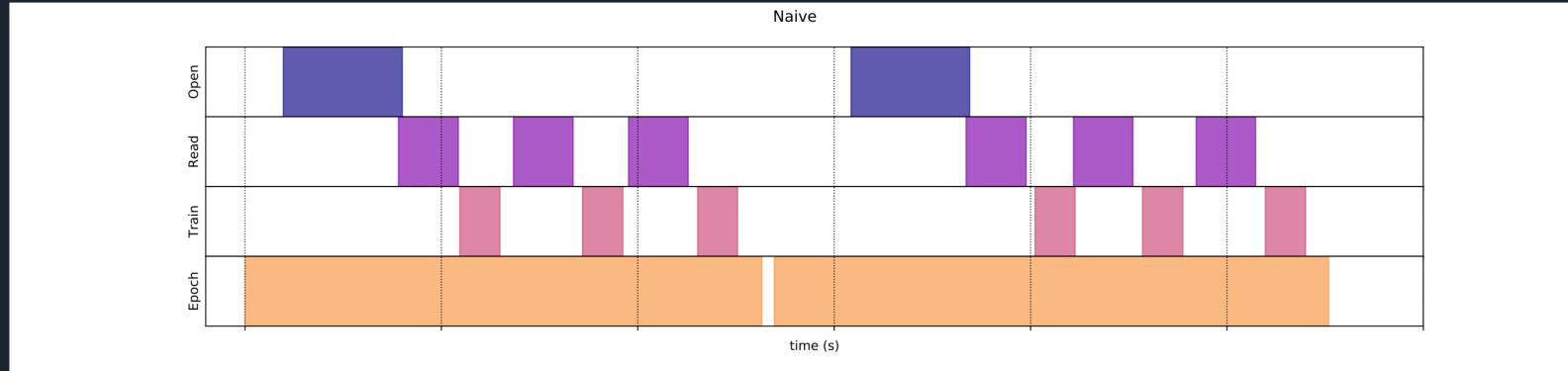


더 이상의 자세한 설명은 생략한다.

<https://arxiv.org/abs/1503.02531>

# CPU도 빡세게 굴리자

- GPU가 열일하는 동안 CPU는 prefetching으로 다음 minibatch를 로드해야 함



- CPU to GPU transfer를 최대한 방지하기



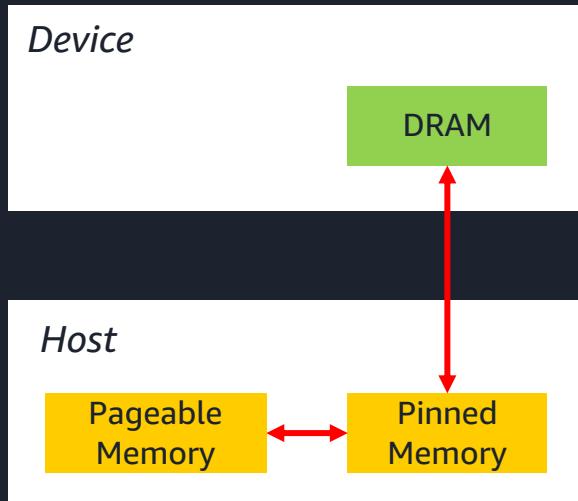
.cpu()  
.item()  
.numpy()



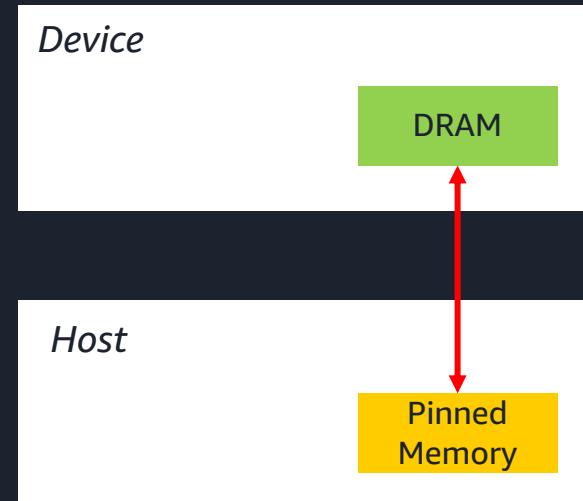
# Pinned Memory (Zero-copy)

- CPU-GPU간 통신은 DMA(Direct Memory Access)
- Host(CPU) 메모리는 Pageable Memory로 DMA 불가 → 복사본 생성 필요

Pageable Data Transfer



Pinned Data Transfer



## 2. Multi GPUs (Single Machine)

# 기본 용어 (준비 운동)

- **rank**: 글로벌 process id (각 GPU는 단일 process에 매칭됨)
- **local\_rank**: 해당 node에서의 process id (a unique local ID for processes running in a single node)
- **node\_size**: 독립된 머신의 수
- **num\_gpu**: 각 머신당 사용할 gpu 개수
- **world\_size**: 총 글로벌 process 개수 ( $\text{node\_size} * \text{num\_gpu}$ )



# 기본 용어 (준비 운동)

- **rank**: 글로벌 process id (각 GPU는 단일 process에 매칭됨)
- **local\_rank**: 해당 node에서의 process id (a unique local ID for processes running in a single node)
- **node\_size**: 독립된 머신의 수
- **num\_gpu**: 각 머신당 사용할 gpu 개수
- **world\_size**: 총 글로벌 process 개수 (node\_size \*num\_gpu)

Local rank



Node 0 (0번 머신)  
GPU 4장



Node 1 (0번 머신)  
GPU 4장

# PyTorch DataParallel

- 사용법이 매우 간단하지만 (Only 1-line code), 1개의 GPU에 리소스가 몰리므로 메모리 불균형 문제 발생
- 또한, multi processing이 아닌 multi threading 방식으로 파이썬 인터프리터의 GIL(Global Interpreter Lock; 다른 thread는 자원을 acquire하기 전까지는 실행 불가능)로 인한 퍼포먼스 오버헤드 발생

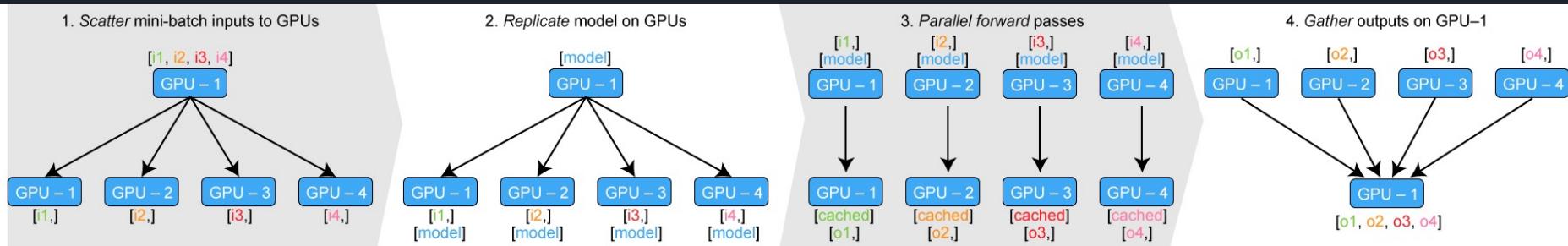
```
parallel_model = torch.nn.DataParallel(model) # Encapsulate the model
predictions = parallel_model(inputs) # Forward pass on multi-GPUs
loss = loss_function(predictions, labels) # Compute loss function
loss.mean().backward() # Average GPU-losses + backward pass
optimizer.step() # Optimizer step
predictions = parallel_model(inputs) # Forward pass with new parameters
```

# PyTorch DataParallel

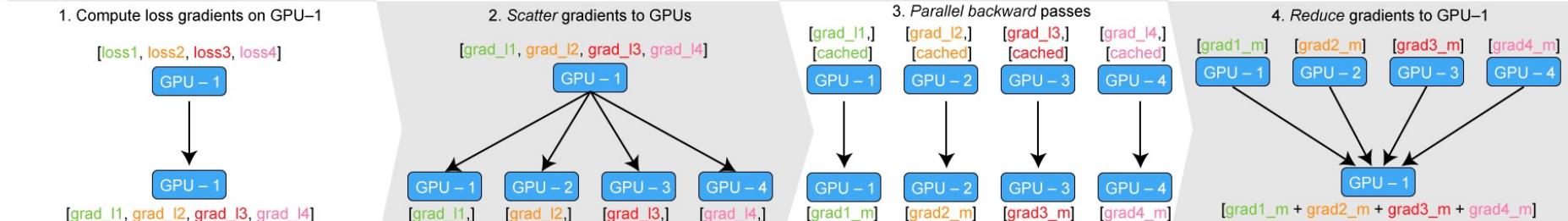
**replicate → scatter → parallel\_apply → gather** 순서로 진행

- **replicate**: 모델을 각 GPU에 복사
- **scatter**: mini-batch를 GPU 개수만큼 나눈 다음 분배
- **parallel\_apply**: Forward propagation
- **gather**: tensor 출력 결과를 하나의 GPU로 모음

Forward



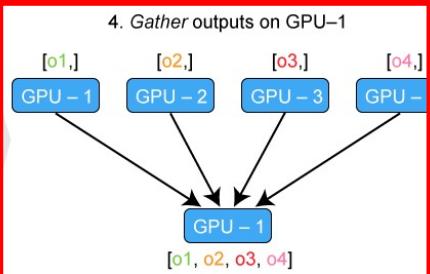
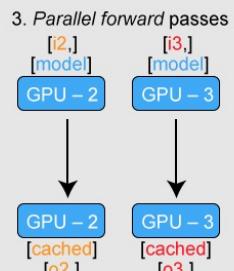
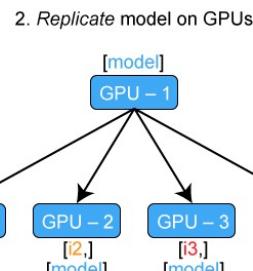
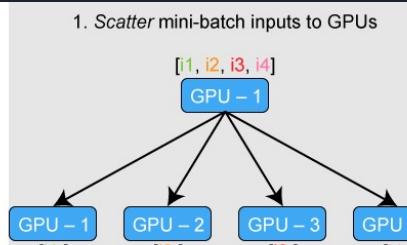
Backward



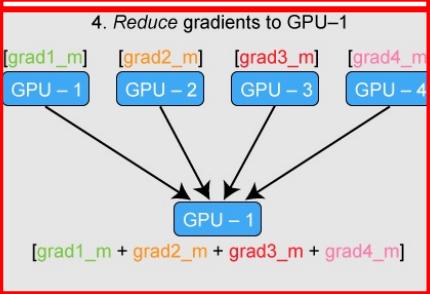
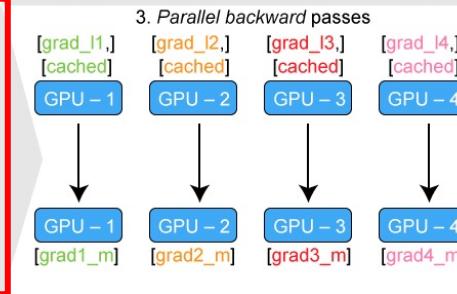
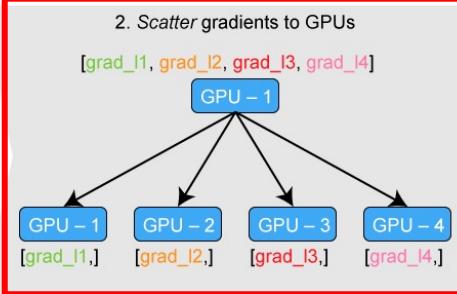
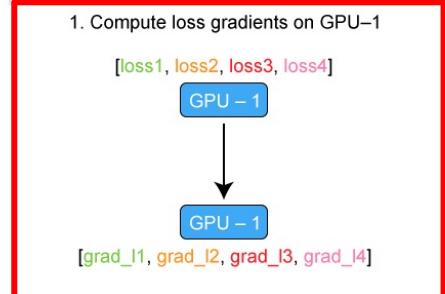
# Quiz

- Question: 어느 부분에서 병목이 발생하나요? 이유는 무엇인가요?
- Easy Question: 몇 번 GPU가 열일하나요?

Forward

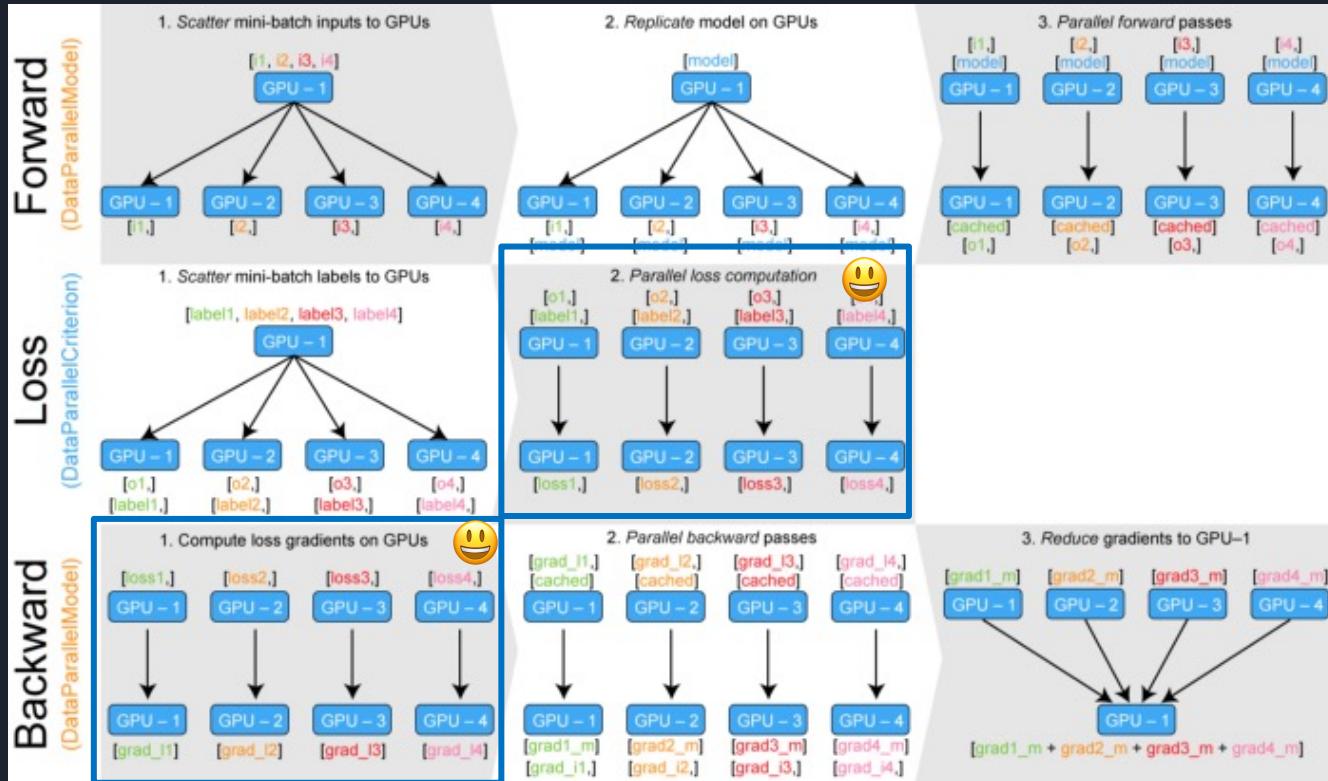


Backward



# Balanced load on multi-GPUs (Custom DataParallel)

- 각 label도 scatter하여 loss function을 parallel하게 연산
- 각 GPU에서 계산한 loss에 대한 backward propagation 수행



# 3. Multi GPUs (Multi Machines)

# Target audience

- 단일 노드에 장착 가능한 최대 GPU 개수: 일반적으로 8
- GPT-3, DALL-E: GPU 1000개 이상 권장



Node 1



Node 2



Node 100

...

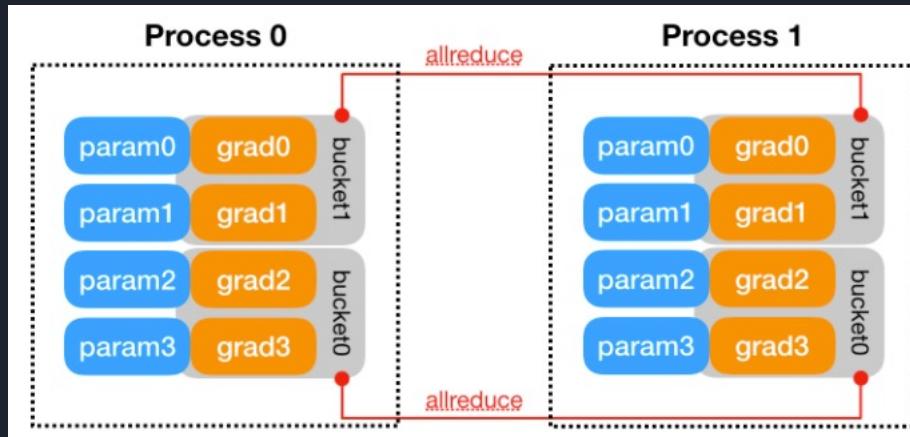
# Left or Right?



- PyTorch DDP (Distributed DataParallel)
- NVIDIA Apex (PyTorch 1.6.0부터 통합됨)
- Uber Horovod
- Amazon SageMaker DDP&DMP: 서울서밋 세션 추천 ^^

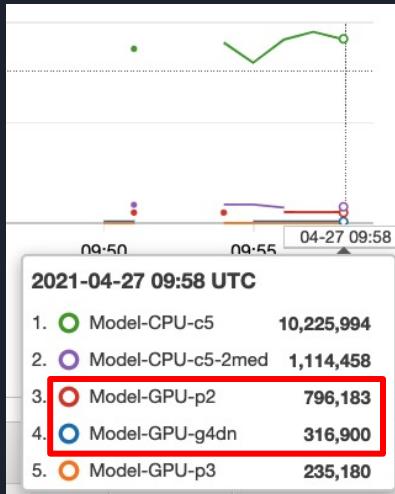
# PyTorch DistributedDataParallel (DDP)

- PyTorch에서 쉽게 사용할 수 있는 multi processing Data Parallelism 기법
- 각 GPU가 dedicated process를 사용하므로 파이썬 인터프리터의 GIL로 인한 퍼포먼스 오버헤드 방지
- `torch.distributed.init_process_group()`을 통해 여러 노드에서 여러 프로세스가 동기화되고 통신할 수 있게 설정
- [https://pytorch.org/tutorials/intermediate/ddp\\_tutorial.html](https://pytorch.org/tutorials/intermediate/ddp_tutorial.html)



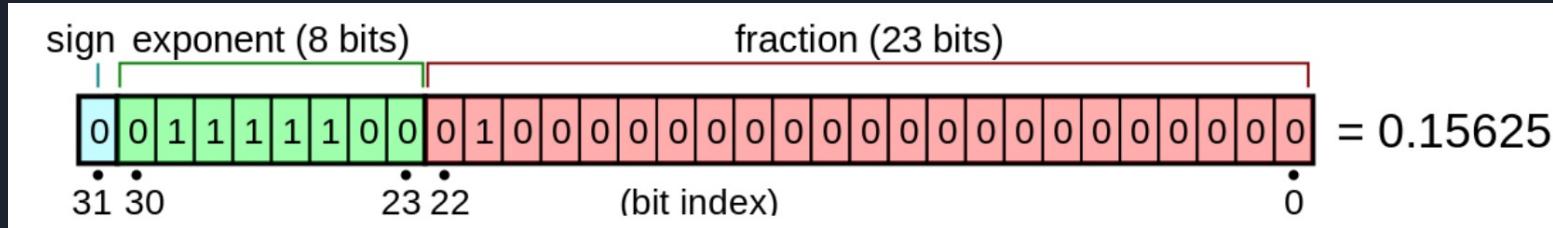
# NVIDIA Apex (A PyTorch Extension)

- 2018년 NVIDIA에서 개발한 분산 학습 PyTorch Extension(APEX = A Pytorch EXtension)으로 PyTorch >= 1.6.0부터 기본 라이브러리로 제공되고 있으며, Automatic Mixed Presicion 기능을 지원함.
- Automatic Mixed Precision: FP16과 FP32를 섞어서 학습
- 3줄의 코드만으로 AMP가 가능하다는 점을 내세우고 있음. 단, Volta 이후의 아키텍처에서 성능 향상이 있으며, p2 계열 인스턴스에서는 권장하지 않음

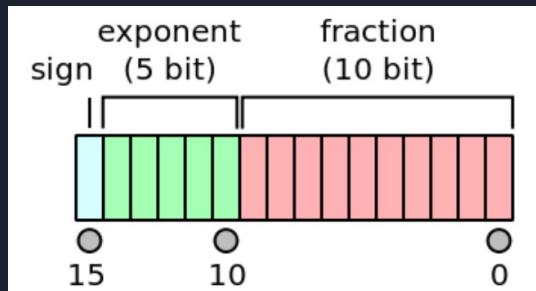


# FP32 and FP16

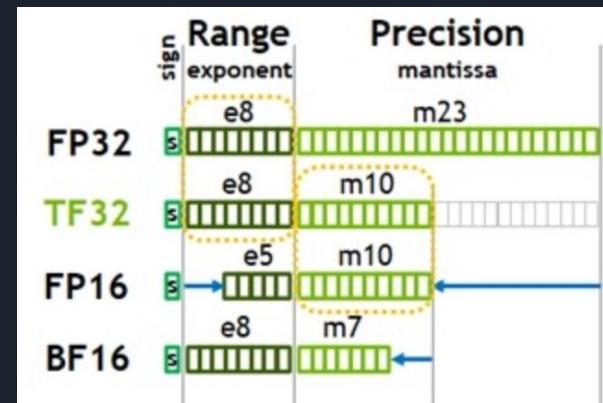
Floating Point를 컴퓨터가 이해할 수 있는 근삿값으로 표현



FP32



FP16



# NVIDIA Apex

## 일반적인 훈련 코드

```
for batch_idx, (inputs, labels) in enumerate(data_loader):
    optimizer.zero_grad()

    outputs = model(inputs)
    loss = criterion(outputs, labels)

    loss.backward()
    optimizer.step()
```

## Apex 적용 후

```
# Creates once at the beginning of training
scaler = torch.cuda.amp.GradScaler()

for data, label in data_iter:
    optimizer.zero_grad()
    # Casts operations to mixed precision
    with torch.cuda.amp.autocast():
        output = model(data)
        loss = loss(output, label)

        # Scales the loss, and create scaled gradients
        scaler.scale(loss).backward()

        # Unscales gradients and calls or skips optimizer.step()
        scaler.step(optimizer)
        scaler.update()
```

# AMP Optimization Options

opt_label	00	01	02	03
cast_model_type	float32	-	float16	float16
patch_torch_functions	False	True	False	False
keep_batchnorm_fp32	-	-	True	False
master_weights	False	-	True	False
loss_scale	1.0	dynamic	dynamic	1.0

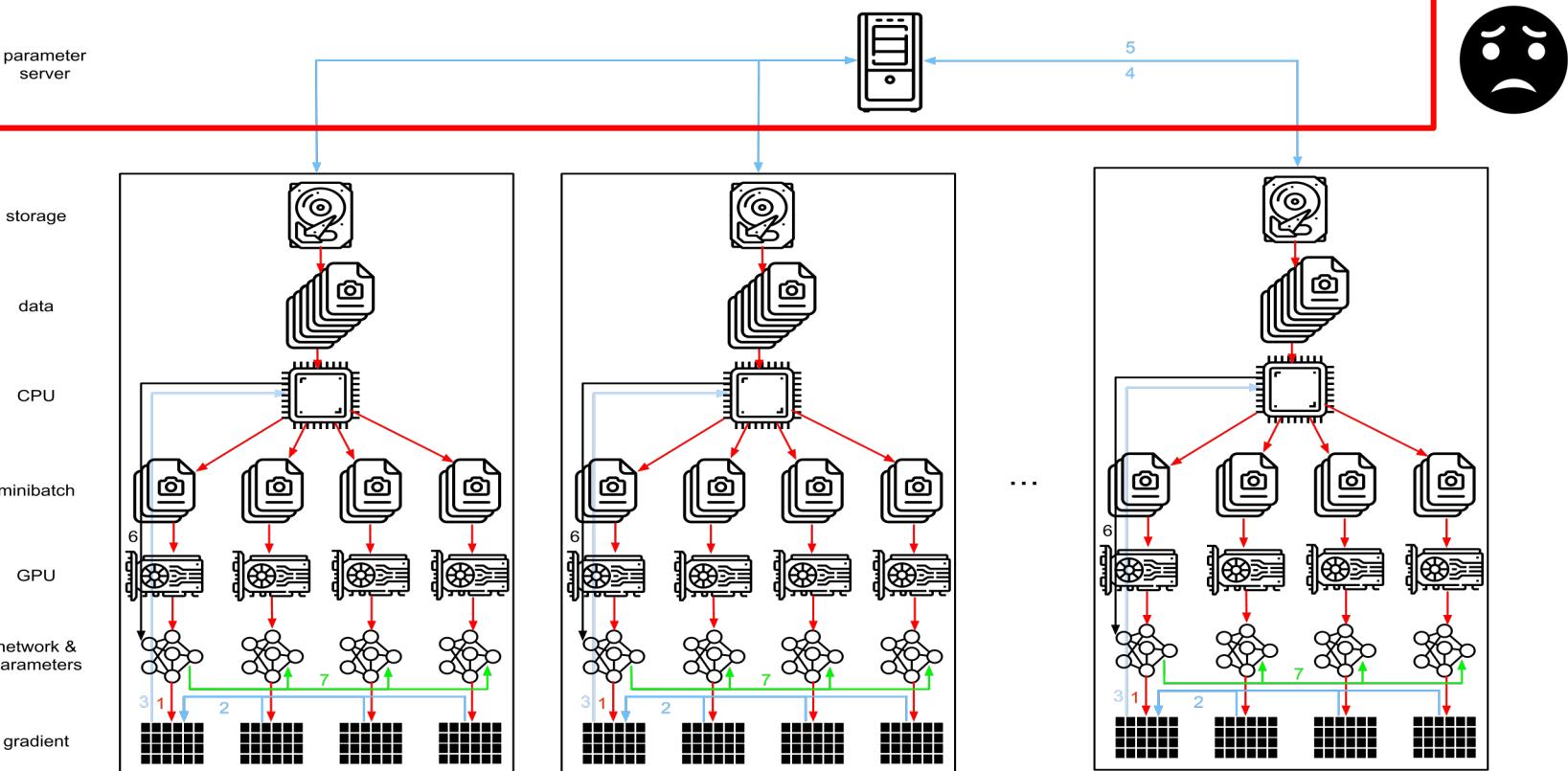
- **opt\_level**

- O0: FP32 training
- O1: [Default] TensorCore을 이용한 FP32 / FP16 혼합 연산으로 TensorCore에 적합한 연산(ops)들은 FP16으로 캐스팅하고 정확한 계산이 필요한 연산들은 FP32를 유지
- O2: Almost FP16 (BatchNorm weight를 제외한 Model weight가 FP16으로 캐스팅)
- O3: FP16 training

- **cast\_model\_type**: 모델 파라메터를 어떤 타입으로 변환할 것인지 여부
- **patch\_torch\_functions**: 함수를 TensorCore용으로 변환할지 여부
- **keep\_batchnorm\_fp32**: BatchNorm 연산을 FP32로 유지할지 여부
- **master\_weights**: 연산 시의 weight를 FP32로 할지 여부
- **loss\_scale**: Gradient Scaling 관련 파라메터

# 4. Collective Communication

# Q1: 파라미터 서버의 병목은?



## Q2: 파라메터 간 통신은 어떻게?

### single- and multi-node communication

#### Message Passing Interface (MPI)

Sets standard + CPU-CPU communication

synchronization, data movement, reduction

#### nVidia Collective Communications Library (nccl)

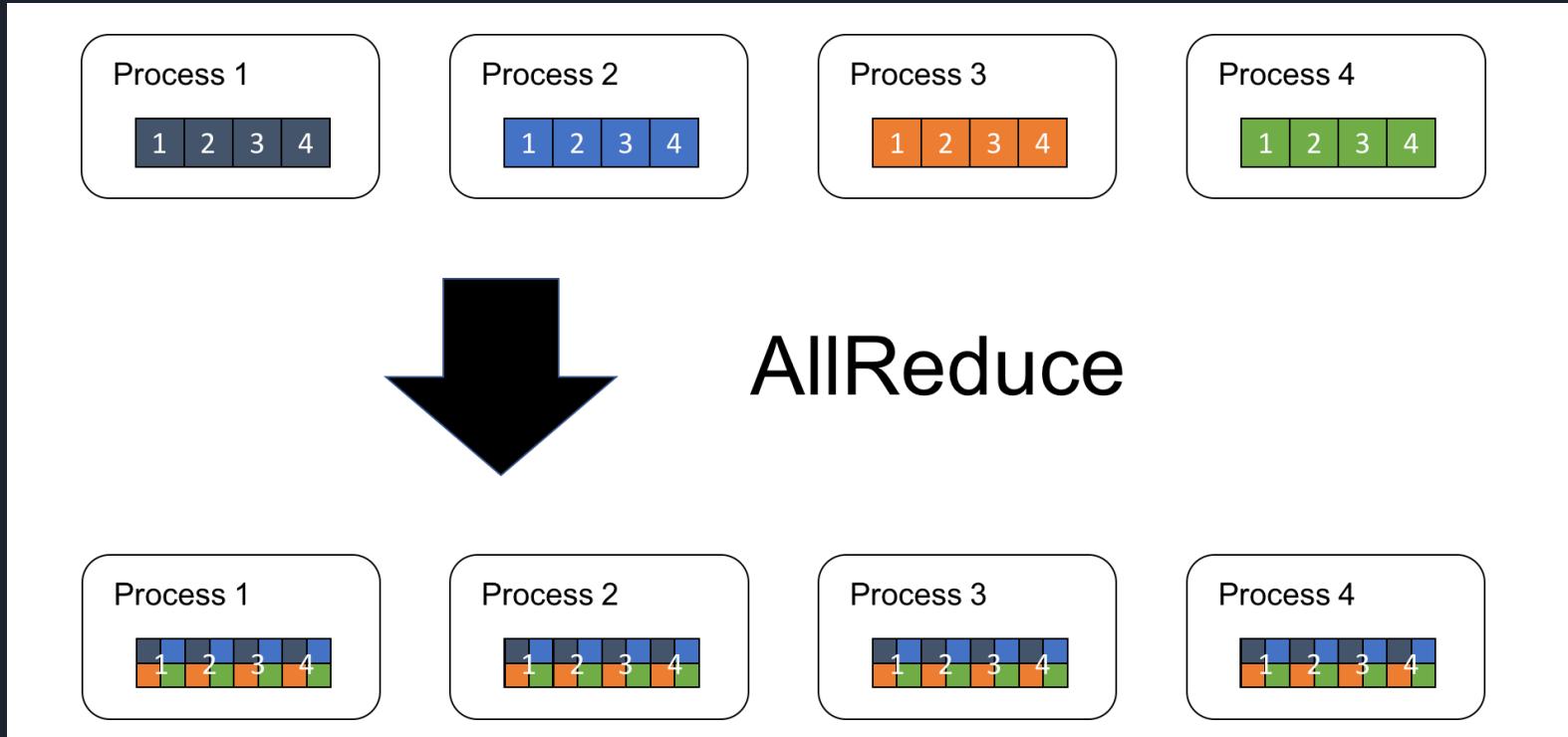
Follows MPI standard for GPU-GPU communication

#### Facebook Gloo

Optimized for ML: CPU-CPU/GPU-GPU communication

# All-Reduce

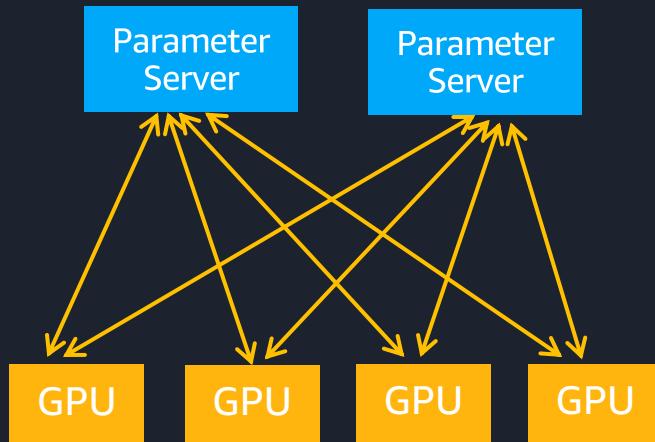
모든 프로세스가 가지고 있는 배열 데이터를 집계 후(Reduce) 집계 결과를 모든 프로세스로 반환



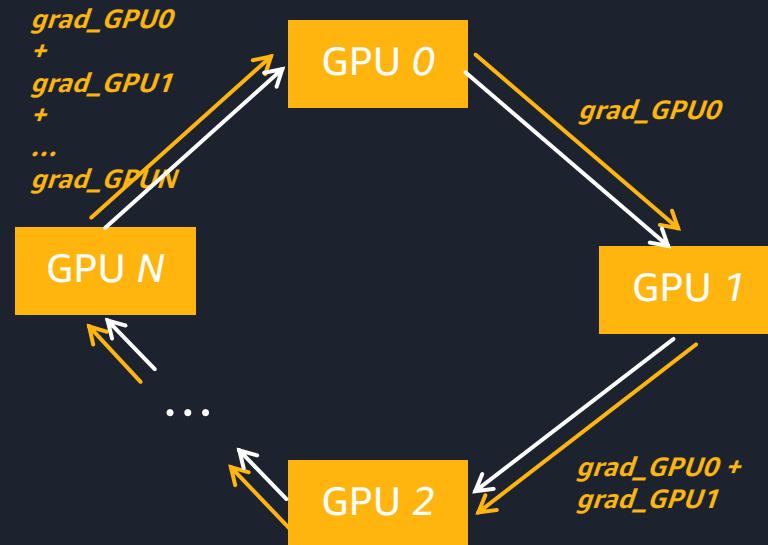
# Ring All-Reduce

- Communication Cost 대폭 개선
- Q: 그럼에도 여전히 문제가 있는데 무엇일까요? (2가지)

Parameter Server



Ring All-reduce (Horovod)

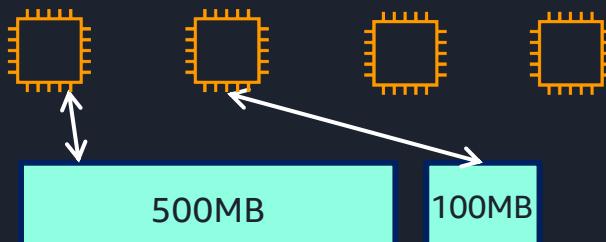


# 그래~서!! SageMaker Data Parallelism 입니다.

핵심 Concept: Balanced Fusion Buffers (BFB)

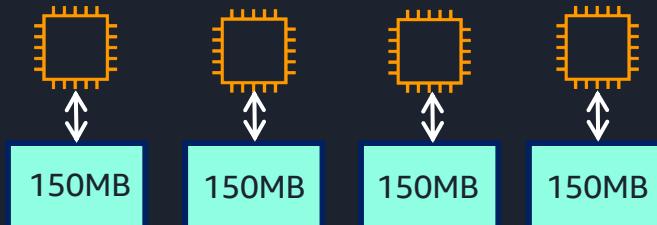
## Traditional

- 전통적인 파라메터 서버는 변수를 원자 단위<sup>atomic unit</sup>로 취급
- 각 변수는 하나의 서버에 배치
- 각 서버는 각 역방향 패스 경로 중 일부만 활성화



## BFB

- BFB: gradient를 보유하는 각 GPU의 버퍼
- 각 서버마다 정확히 동일한 바이트를 할당
- i번째 서버는 모든 worker로부터 BFB의 i 번째 파티션을 수신하고 이를 합산하여 결과를 모든 worker에게 전송



파라메터 서버: Worker = 1:1

# 5. Hardware Communication

# AWS GPU instances

- g4 인스턴스 탑재는 T4(Turing 아키텍처) GPU로 NVLink 미지원 (PCIe Gen3 지원)
- p3 인스턴스 탑재는 V100(Volta 아키텍처) GPU로 NVLink 지원 (EFA 적용 시 100Gbps까지 가능)
  - GPU간 통신 성능: 양방향 25GB/s 대역폭으로 6개의 link를 사용하므로,  $25 \times 6 = 150\text{GB/s}$ 의 대역폭을 지님
- p4 인스턴스 탑재는 A100(Ampere 아키텍처) GPU로 NVSwitch, EFA 및 GPUDirect RDMA를 지원(400Gbps)하므로, CPU를 우회하여 노드 간 GPU-GPU 통신으로 latency 개선 가능
  - A100은 PCIe gen4 x 16를 지원하지만, P4d 인스턴스는 PCIe gen3 x 16으로 제한되어 있음
  - GPU간 통신 성능: 양방향 50GB/s 대역폭으로 12개의 link를 사용하므로  $50 \times 12 = 600\text{GB/s}$ 의 대역폭을 지님

# AWS p4d.24xlarge vs. NVIDIA DGX A100

	AWS p4d.24xlarge	NVIDIA DGX A100
CPU	Intel Xeon Platinum 8275CL x2 sockets = 48 physical cores	AMD EPYC 7742 x2 sockets = 128 physical cores
CPU/GPU 간의 통신	PCIe gen3 x16: 단방향으로 15.75 GB/s	PCIe gen4 x16: 단방향으로 31.51 GB/s
통신 연결 방식	Elastic Fabric Adapter (EFA): <a href="https://aws.amazon.com/ko/hpc/efa/">https://aws.amazon.com/ko/hpc/efa/</a>	ConnextX-6 HDR 200GB/s Infiniband: <a href="https://www.nvidia.com/en-us/networking/infiniband-adapters/connectx-6/">https://www.nvidia.com/en-us/networking/infiniband-adapters/connectx-6/</a>
노드 간 통신 성능	100 Gbps x4 links = 400 Gbps	200 Gbps x8 links = 1600 Gbps
각 노드의 GPU간 통신 성능	600 Gbps	600 Gbps

# Good to know

- Turing 아키텍처 이전까지는 동일한 아키텍처라 하더라도 일부 게이밍 카드에서 FP16 및 FP64 성능이 심각하게 낮음
  - GeForce GTX 1080 Ti: < 0.177 TFLOPS (FP16)
  - Tesla P100: 18.7 ~ 21.2 TFLOPS (FP16)
- Turing 아키텍처부터는 FP16에 대한 성능 제약이 없음
- Profit!! (GeForce RTX 구입 가즈아~)



# References

- Training Deep Nets with Sublinear Memory Cost: <https://arxiv.org/pdf/1604.06174.pdf>
- Fitting larger networks into memory: <https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9>
- PyTorch Multi-GPU 제대로 학습하기(당근마켓 블로그): <https://medium.com/daanqn/pytorch-multi-gpu-%ED%95%99%EC%8A%B5-%EC%A0%9C%EB%8C%80%EB%A1%9C-%ED%95%98%EA%B8%B0-27270617936b>
- Technologies behind Distributed Deep Learning: AllReduce:  
<https://tech.preferred.jp/en/blog/technologies-behind-distributed-deep-learning-allreduce/>
- Comparison of NVIDIA Tesla/Quadro and NVIDIA GeForce GPUs:  
<https://www.microway.com/knowledge-center-articles/comparison-of-nvidia-geforce-gpus-and-nvidia-tesla-gpus/>
- OpenMPI: <https://www.open-mpi.org/>
- NVIDIA NCCL: <https://developer.nvidia.com/nccl>
- PyTorch Official Documents: <https://pytorch.org/docs/stable/distributed.html>

# 감사합니다.