

## ▼ (01강) Introduction to PyTorch

### ▼ 딥러닝 오픈소스 프레임워크




Tensorflow : 구글에서 문서화를 위해 만듦

PyTorch : 페이스북이 만듦

CNTK : 마이크로소프트이 만듦

mxnet : 아마존이 만듦

- Duopoly : 구글과 페이스북의 디지털 마케팅 분야 복점

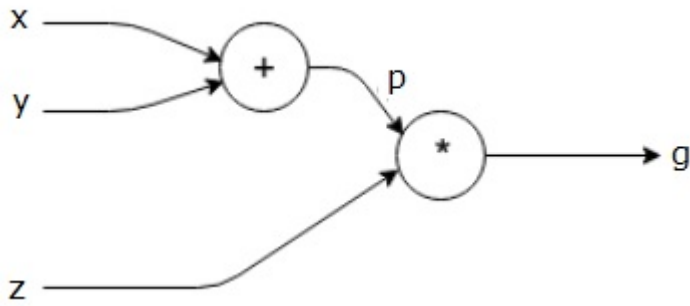
	Keras 	TensorFlow 	PyTorch 
Level of API	high-level API <sup>1</sup>	Both high & low level APIs	Lower-level API <sup>2</sup>
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex <sup>3</sup>
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook <sup>4</sup>
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs <sup>5</sup>

이미지 출처 : <https://datasciencecareer.wordpress.com/2020/12/09/ml03-pytorch-vs-tensorflow/>

### ▼ Computational Graph

연산의 과정을 그래프로 표현

$$g = (x + y) * z$$



- TF - Define and Run : 그래프를 먼저 정의하고 실행시점에 데이터 feed
- PyTorch - Define by Run (Dynamic Computational Graph, DCG) : 실행을 하면서 그래프를 생성하는 방식, 디버그 용이

## TF 장점

- TF는 production 과 scalability의 장점
- 클라우드 연결
- 멀티 GPU, TPU 지원

## PyTorch의 장점

- Define by Run 의 장점 : 즉시 확인 가능 → pythonic code(핵심)
- GPU support, Good API and community
- 사용하기 편한 장점이 가장 큼

## PyTorch 구조

PyTorch : Numpy + AutoGrad + Function

- Numpy 구조를 가지는 Tensor 객체로 array를 표현
- 자동미분을 지원하여 딥러닝 연산을 지원
- 다양한 형태의 딥러닝을 지원하는 함수와 모델을 지원함

## ▼ (02강) PyTorch Basics

### Tensor

- 다차원 Arrays 를 표현하는 PyTorch 클래스
- 사실상 numpy의 ndarray와 동일 (그러므로 TensorFlow의 Tensor와도 동일)
- Tensor를 생성하는 함수도 거의 동일
- 기본적으로 tensor가 가질수 있는 자료형은 numpy와 동일하나 GPU 사용 가능 여부만 다름

- pytorch의 tensor는 GPU에 올려서 사용가능

## 실습 코드

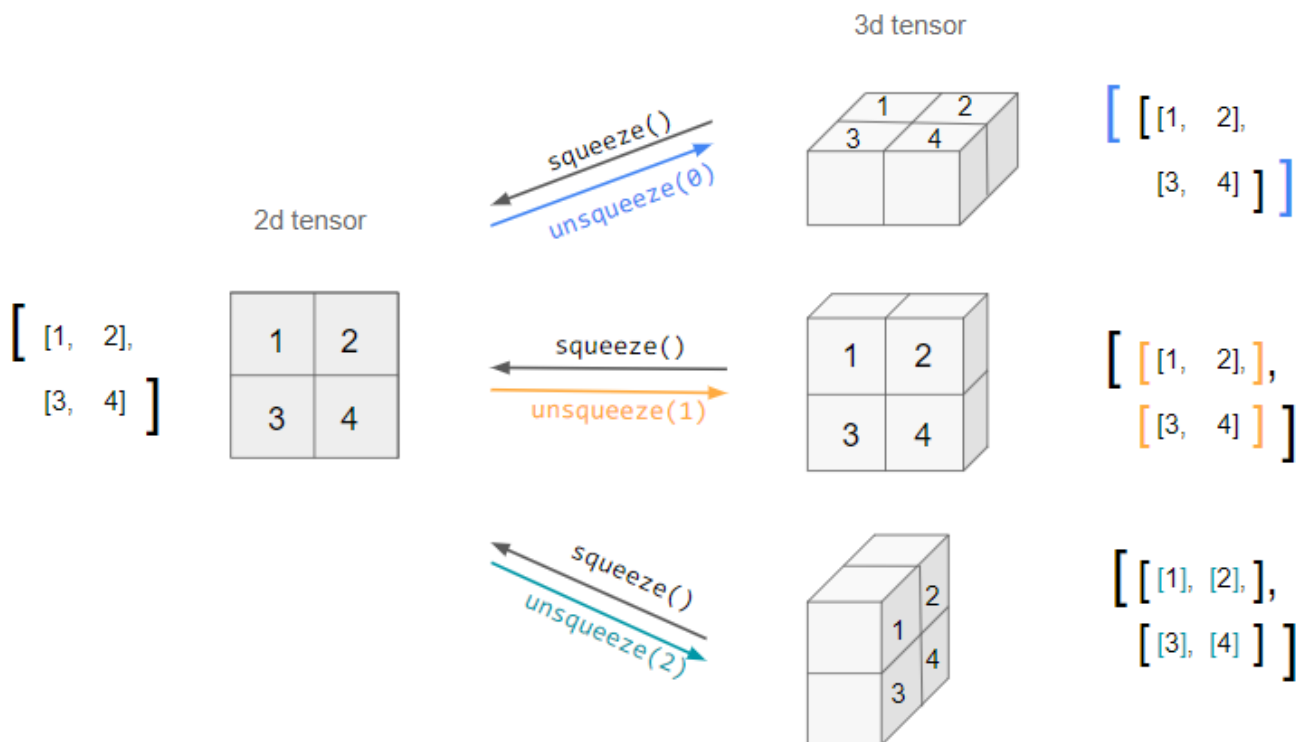
[ ] ↳ 숨겨진 셀 10개

## Tensor handling

view, squeeze, unsqueeze 등으로 tensor 조정가능

- view: reshape과 동일하게 tensor의 shape을 변환
- squeeze: 차원의 개수가 1인 차원을 삭제 (압축)
- unsqueeze: 차원의 개수가 1인 차원을 추가

view와 reshape은 contiguity 보장의 차이



이미지 출처 : <https://stackoverflow.com/questions/61598771/pytorch-squeeze-and-unsqueeze>

## 실습 코드

[ ] ↳ 숨겨진 셀 14개

## Tensor operations

- 기본적인 tensor의 operations는 numpy와 동일
- 행렬곱셈 연산은 함수는 dot이 아닌 mm 사용 : 벡터 연산과 행렬 연산 구분
- mm과 matmul은 broadcasting 지원 차리

### 실습 코드

[ ] ↳ 숨겨진 셀 17개

## Tensor operations for ML/DL formula

- nn.functional 모듈을 통해 다양한 수식 변환을 지원함

### 실습 코드

[ ] ↳ 숨겨진 셀 14개

## AutoGrad

- PyTorch의 핵심은 backward 함수로 자동 미분의 지원하는 것
- AutoGrad 튜토리얼
  - [https://pytorch.org/tutorials/beginner/blitz/autograd\\_tutorial.html](https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html)
- Tensor와 AutoGrad 튜토리얼
  - [https://pytorch.org/tutorials/beginner/examples\\_autograd/two\\_layer\\_net\\_autograd.html](https://pytorch.org/tutorials/beginner/examples_autograd/two_layer_net_autograd.html)

### 실습 코드

[ ] ↳ 숨겨진 셀 8개

## ▼ (03강) PyTorch 프로젝트 구조 이해하기

### PyTorch Project Template 참고 자료

Pytorch Template : <https://github.com/victoresque/pytorch-template>

Pytorch Template 2 : <https://github.com/FrancescoSaverioZuppichini/PyTorch-Deep-Learning-Template>

Pytorch Lightning Template : <https://github.com/PyTorchLightning/deep-learning-project-template>

Pytorch Lightning : <https://www.pytorchlightning.ai/>

Pytorch Lightning + NNI Boilerplate : <https://github.com/davinnovation/pytorch-boilerplate>

## ▶ PyTorch Project Template Overview

↳ 숨겨진 셀 1개

## ✓ ▶ 템플릿 실습 코드

[ ] ↳ 숨겨진 셀 5개

## ! ▶ ngrock 실습 코드

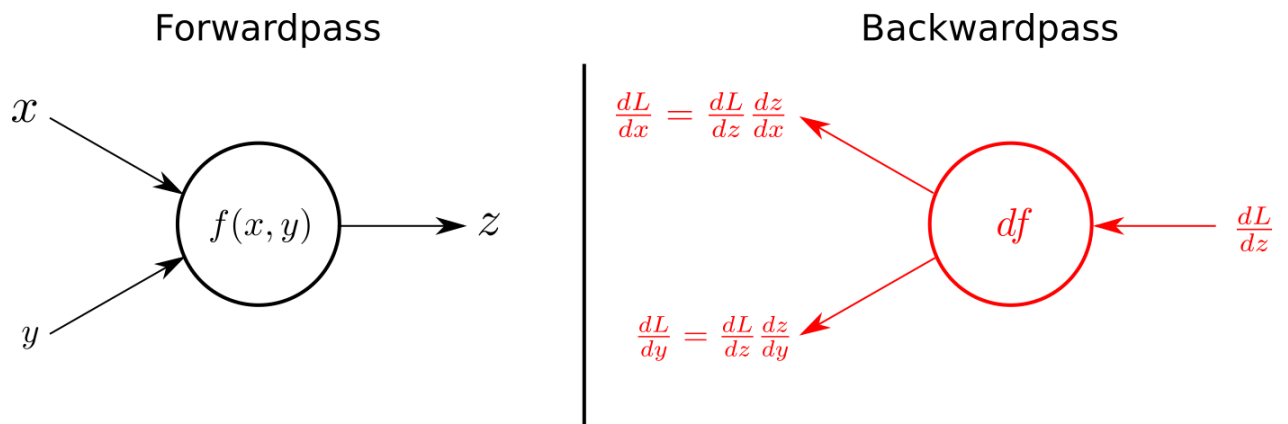
[ ] ↳ 숨겨진 셀 10개

## ▼ (04강) AutoGrad & Optimizer

- 레이어 == 레고 블록
- 딥러닝 == 레이어의 반복 연속
- Pytorch로 Linear Regression하기
  - <https://towardsdatascience.com/linear-regression-with-pytorch-eb6dedead817>
- Pytorch로 Logistic Regression하기
  - <https://medium.com/dair-ai/implementing-a-logistic-regression-model-from-scratch-with-pytorch-24ea062cd856>

## ▼ torch.nn.Module

- torch.nn.Module 정의 : 딥러닝을 구성하는 Layer의 base class
- 필수 정의 요소 : Input, Output, Forward, Backward(AutoGrad) 정의
- 학습의 대상이 되는 parameter(tensor; weight) 정의



이미지 출처 : <https://github.com/Vercaca/NN-Backpropagation>

## nn.Parameter

- Tensor 객체의 상속 객체
- nn.Module 내에 **attribute**가 될 때는 **required\_grad=True** 로 지정되어 학습 대상(AutoGrad의 대상)이 되는 Tensor
- 우리가 직접 지정할 일은 잘 없음 : 대부분의 layer에는 weights 값들이 지정되어 있음

### ✓ 코드 실습

[ ] ↳ 숨겨진 셀 10개

## Backward

- Layer에 있는 Parameter들의 미분을 수행
- Forward의 결과값 (model의 output=예측치)과 실제값간의 차이(loss) 에 대해 미분을 수행
  - 미분값은 AutoGrad로 계산된다.
- AutoGrad의 결과 값으로 Parameter 업데이트
- forward는  $y_{\text{hat}}$ 을 구하는 것

### ✓ 코드 실습

[ ] ↳ 숨겨진 셀 13개

## Backward from the scratch

- 실제 backward는 Module 단계에서 직접 지정 가능

- nn.Module에서 backward 와 optimize 오버라이딩
- 사용자가 직접 미분 수식을 써야하기에 부담
  - 쓸일은 없으나 순서는 이해는 해두자.

Dataloader : 데이터를 가져와서 GPU에 feeding

## ✓ ▶ 코드 실습

[ ] ↳ 숨겨진 셀 15개

## ▼ (05강) Dataset & Dataloader

### Dataset 클래스

- 데이터 입력 형태를 정의하는 클래스
- 데이터를 입력하는 방식의 표준화
- Image, Text, Audio 등에 따른 다른 입력정의

### Dataset 클래스 생성시 유의점

- 데이터 형태에 따라 각 함수를 다르게 정의
- 모든 것을 데이터 생성 시점에 처리할 필요는 없음
  - image의 Tensor 변화는 학습에 **필요한 시점에 변환**
- 데이터 셋에 대한 표준화된 처리방법 제공 필요
- 최근에는 HuggingFace등 표준화된 라이브러리 사용

### Dataloader 클래스

- Data의 Batch를 생성해주는 클래스
- 학습 직전(CPU -> GPU feed 전) 데이터의 변환을 책임
- Tensor로 변환 + Batch 처리가 메인 업무
- 병렬적인 데이터 전처리 코드의 고민 필요

### Pytorch Dataset, Dataloader 튜토리얼

- [https://pytorch.org/tutorials/beginner/basics/data\\_tutorial.html](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

## ✓ ▶ 코드 실습

[ ] ↳ 숨겨진 셀 19개

## ▼ (06강) 모델 불러오기

모델 레포

- CV : <https://github.com/rwightman/pytorch-image-models>
- Segmentation : [https://github.com/qubvel/segmentation\\_models.pytorch](https://github.com/qubvel/segmentation_models.pytorch)

### model.save()

- 학습의 결과를 저장하기 위한 함수
  - 저장하는 것 : 모델 형태(architecture)와 파라미터(용량 작음)를 저장
- 모델 학습 중간 과정의 저장을 통해 최선의 결과모델을 선택
  - Early stoping 쓸 때
- 만들어진 모델을 외부 연구자와 공유하여 학습 재연성 향상

## ✓ 코드 실습

[ ] ↳ 숨겨진 셀 16개

## ▼ checkpoints

- 학습의 중간 결과를 저장하여 **최선의 결과를 선택**
- earlystopping 기법 사용시 이전 학습의 결과물을 저장
- loss와 metric 값을 지속적으로 확인 저장
- 일반적으로 epoch, loss, metric을 함께 저장하여 확인
- 런타임이 끊기는 colab에서 지속적인 학습을 위해 필요

## ! 코드 실습

[ ] ↳ 숨겨진 셀 12개

## Transfer learning

- 다른 대용량 데이터셋으로 만든 모델을 현재 데이터에 적용
- 일반적으로 대용량 데이터셋으로 만들어진 모델의 성능 향상
- 현재의 딥러닝에서 가장 일반적인 학습 기법
- backbone architecture가 잘 학습된 모델에서 일부분만 변경하여 학습을 수행함



## 주의사항

- .pth보다 .pt를 사용하자
- 보통 기존 모델에 새로운 레이어를 추가하는 편이다.

## 모델 링크

### 1. PyTorch Image Models

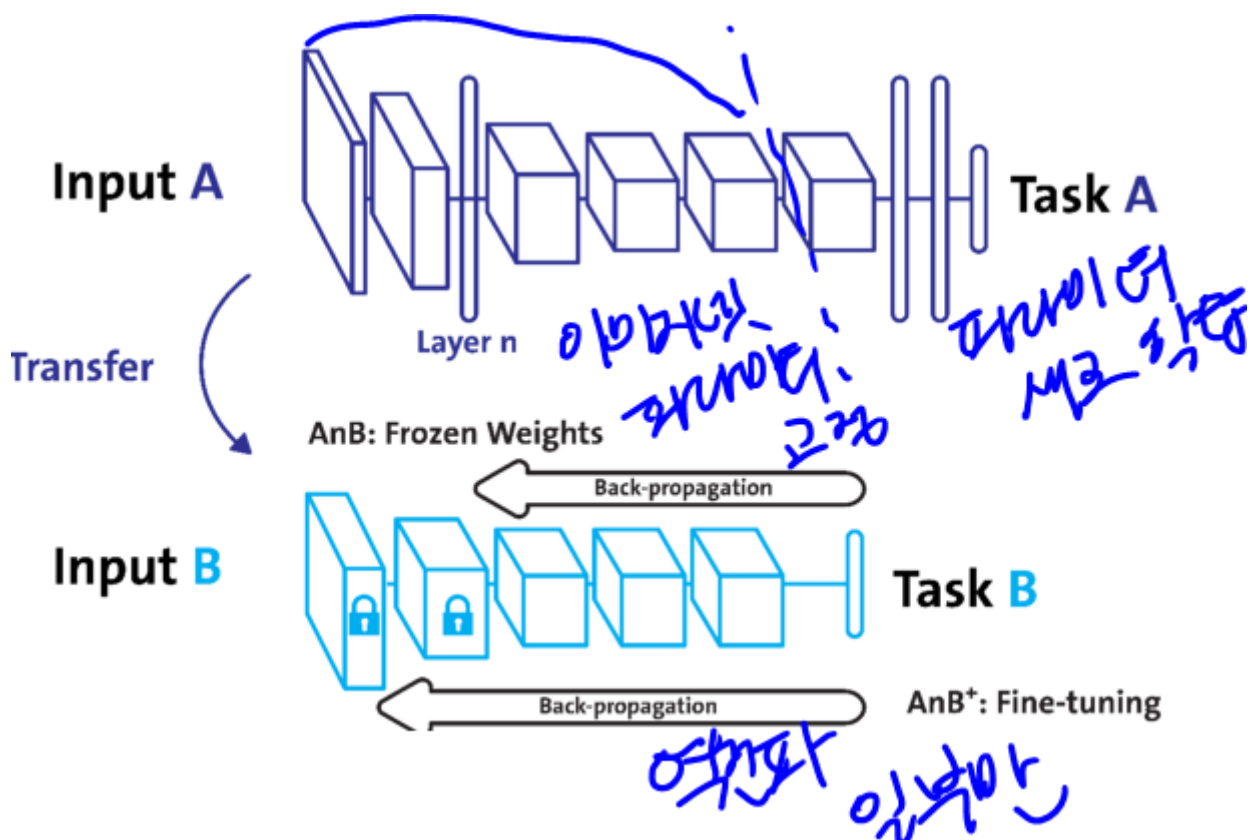
- <https://github.com/rwightman/pytorch-image-models#introduction>

### 2. HuggingFace

- <https://huggingface.co/models>

## ▼ Freezing

- 단계적 Freezing도 사용



이미지 출처 : <https://purnasaigudikandula.medium.com/deep-view-on-transfer-learning-with-image-classification-pytorch-5cf963939575>

## ✓ 코드 실습

[ ] ↳ 숨겨진 셀 29개

## ▼ (07강) Monitoring tools for PyTorch

### Tensorboard

- TensorFlow의 프로젝트로 만들어진 시각화 도구
- 학습 그래프, metric, 학습 결과의 시각화 지원
- **PyTorch도 연결 가능, 딥러닝 시각화 핵심 도구**

### Tensorboard에서 저장가능한 값

- scalar : metric(acc, Loss) 등 상수 값의 **연속(epoch)**을 표시
- graph : 모델의 computational graph 표시
- histogram : weight 등 값의 분포를 표현, weight의 분포는 정규분포면 좋음
- Image/Text : 예측 값과 실제 값을 비교 표시
- mesh : 3d 형태의 데이터를 표현하는 도구
- add\_hparams : 하이퍼 파라미터 기록

scalar, graph, histogram, Image/Text 많이 사용

### 레퍼런스

- WandB 모니터링 도구
  - <https://wandb.ai/site>
- Pytorch Tensorboard
  - <https://pytorch.org/docs/stable/tensorboard.html>
- Pytorch Lightning Logger 목록들
  - <https://pytorch-lightning.readthedocs.io/en/stable/extensions/logging.html>

## ✓ ▶ 실습 코드

[ ] ↳ 숨겨진 셀 32개

### weight & biases

- 머신러닝 실험을 원활히 지원하는 상용도구
- 협업, code versioning, 실험 결과 기록 등 제공
- MLOps의 대표적인 툴이 되고 있음
- 기본 기능 무료

## ✓ ▶ 실습 코드

## ▼ (08강) Multi-GPU 학습

### 개념

- Single vs. Multi
  - Single : GPU 1개
  - Multi : GPU 2개 이상
- GPU vs. Node
  - GPU :
  - Node(System) : 1대의 컴퓨터
- Single Node Single GPU : 1대의 컴퓨터, 1개의 GPU
- Single Node Multi GPU : 1대의 컴퓨터, 2개 이상 의 GPU
- Multi Node Multi GPU : 2대 이상의 컴퓨터, 2개 이상 의 GPU
- TensorRT : 딥러닝 추론 최적화 라이브러리

### 참고자료

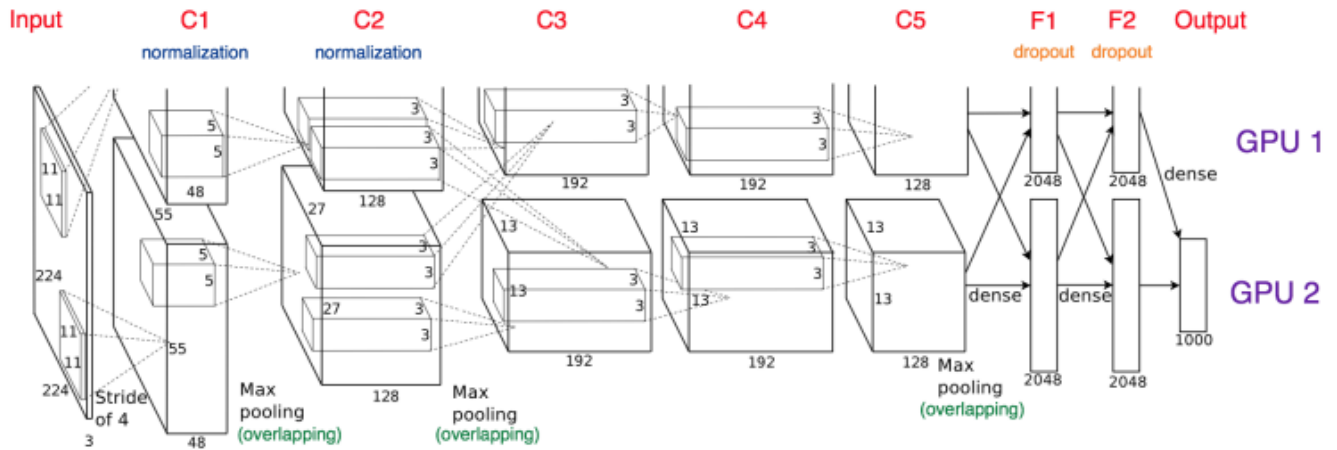
- Pytorch Lightning Multi GPU 학습 : [https://pytorch-lightning.readthedocs.io/en/stable/advanced/multi\\_gpu.html](https://pytorch-lightning.readthedocs.io/en/stable/advanced/multi_gpu.html)
- DDP 튜토리얼 : [https://pytorch.org/tutorials/intermediate/ddp\\_tutorial.html](https://pytorch.org/tutorials/intermediate/ddp_tutorial.html)

## ▼ Model parallel

- 다중 GPU에 학습을 분산하는 두 가지 방법
  - 모델 병렬화 : 모델을 나누기
  - 데이터 병렬화 : 데이터를 나누기
- 모델을 나누는 것은 생각보다 예전부터 썼음 (Alexnet)
- 모델 병렬화는 모델의 병목, 파이프라인의 어려움이 발생하는 고난이도 과제이고 흔하게 사용하지는 않는다.

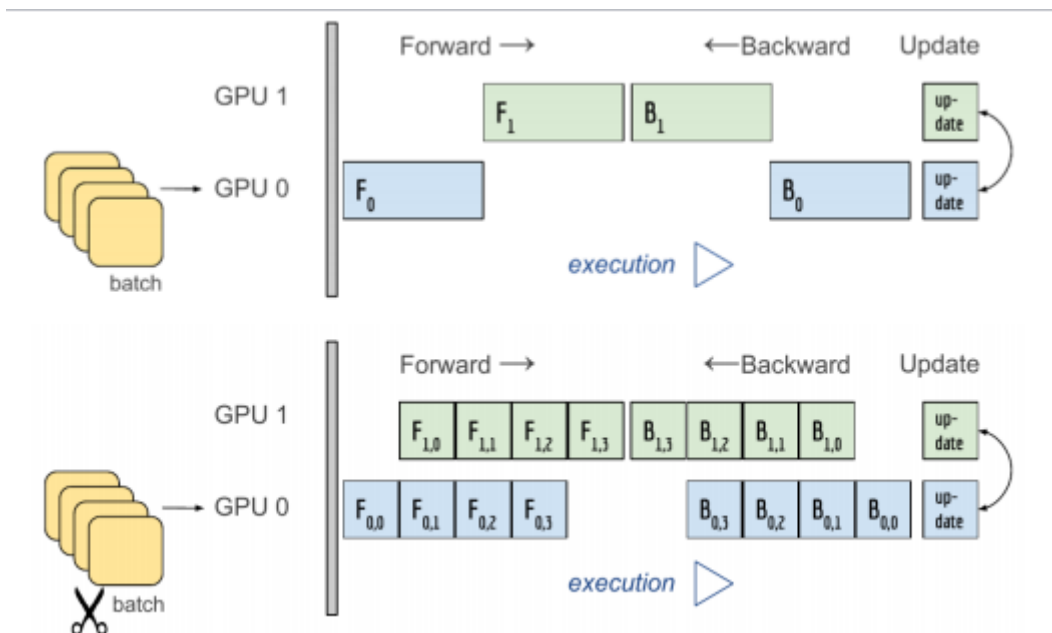
## ▼ 모델 나누기

예시 : AlexNet, GPU 메모리가 작아서



좋은 병렬화는 배치처리를 잘 해서 양 GPU 모두 계속 사용할 수 있어야 한다.

이미지 출처 : <http://www.idris.fr/eng/ia/model-parallelism-pytorch-eng.html>



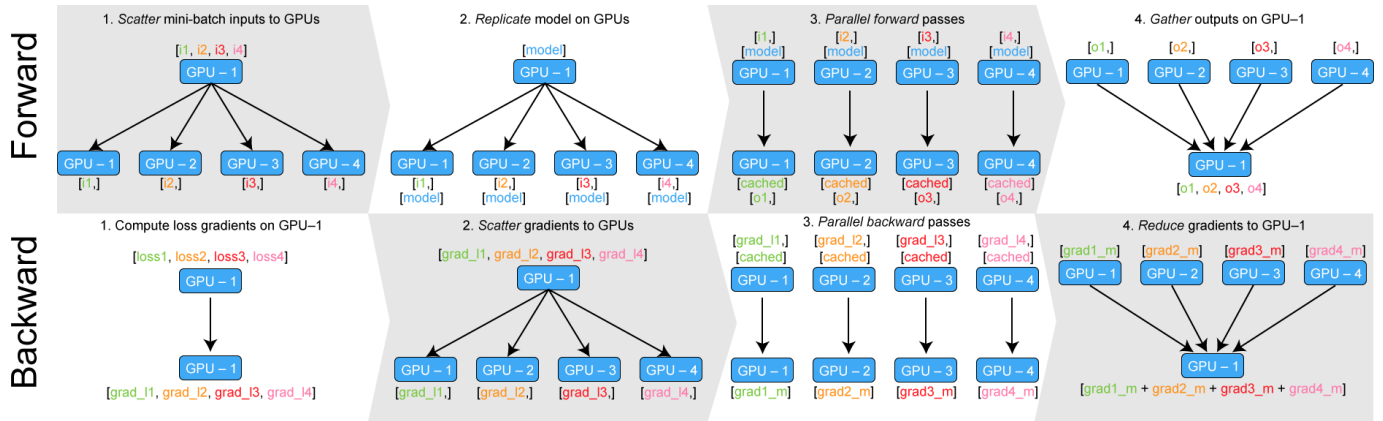
## ▶ 실습 코드

↳ 숨겨진 셀 1개

## ▼ 데이터 나누기

- 데이터를 나눠 GPU에 할당후 결과의 평균을 취하는 방법
- Minibatch 수식과 유사, 한번에 여러 GPU에서 수행

아래 이미지 출처 : <https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255>



## PyTorch의 데이터 병렬화

### DataParallel : 단순히 데이터를 분배한후 평균을 취함

- GIL (Global interpreter lock) 문제 발생 : 하나의 GPU가 취합해서 GPU 사용 불균형 문제 발생, 해당 GPU 메모리에 맞춰서 Batch 사이즈 감소 (한 GPU가 병목)
- 파이토치 함수로 쉽게 구현 가능

### DistributedDataParallel : 각 CPU마다 process 생성하여 개별 GPU에 할당

- 취합하는 작업 없음 : 기본적으로 DataParallel로 하나 개별적으로 연산의 평균을 냄

## 실습 코드

↳ 숨겨진 셀 3개

## (09강) Hyperparameter Tuning

### 성능 향상 방법

1. 모델 수정 : 이미 모델을 많이 있음
2. 데이터 추가 : 가장 효과가 좋음
3. 초매개변수 조정

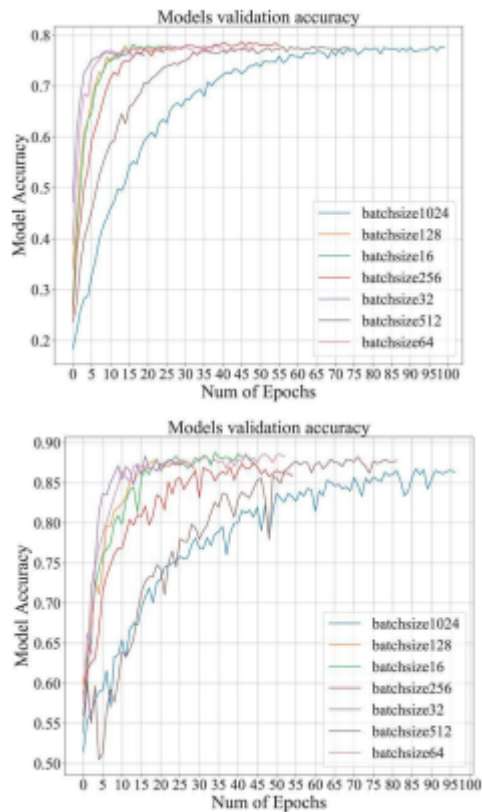
### 정의

- 모델 스스로 학습하지 않는 값은 사람이 지정
  - learning rate : NAS, AutoML
  - 모델의 크기
  - optimizer
- 하이퍼 파라미터에 의해서 값의 크게 좌우 될 때도 있었으나 요즘에는 드물다.
  - 2015~2016 구글의 레시피 따라야 했다.
  - 요즘에는 데이터가 많고 모델 성능이 좋아서 드물다.

- 스코어 0.01 올리기 위해 도전

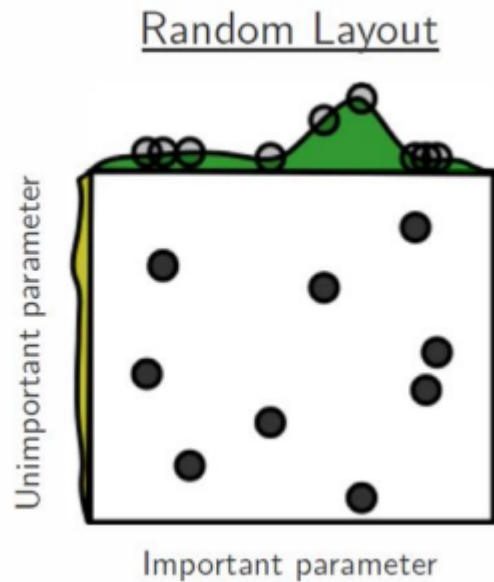
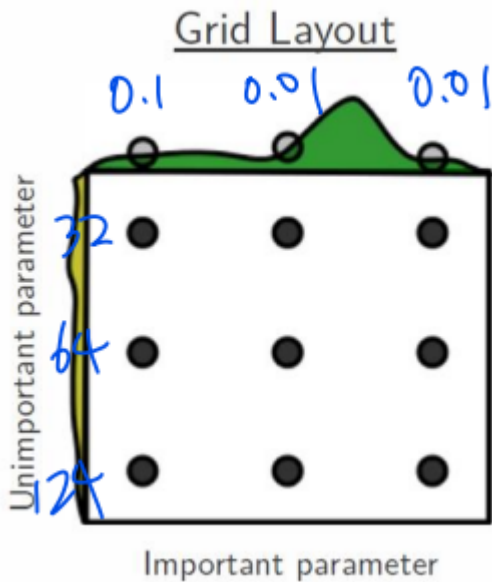
참고자료 Pytorch와 Ray 같이 사용하기 :

[https://pytorch.org/tutorials/beginner/hyperparameter\\_tuning\\_tutorial.html](https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html)



이미지 출처 : <https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs190033>


- 가장 기본적인 방법 – grid vs random
  - 로그 스케일로 변화 : 0.1 -> 0.01 -> 0.001
- 랜덤 서치하다가 학습 잘 되는 구간에서 그리드 서치 실행
- 최근에는 베이지안 기반 기법들이 주도 (BOHB, 2018)



## ▼ Ray

- multi-node multi processing 지원 모듈
- ML/DL의 병렬 처리를 위해 개발된 모듈
- 현재 분산병렬 ML/DL 모듈의 표준
- Hyperparameter Search에 사용되는 다양한 모듈 제공

## ▶ 실습 코드

 숨겨진 셀 11개

## ▼ (10강) PyTorch Troubleshooting

### OOM가 발생하면

- OOM이 왜 발생했는지 알기 어려움
- OOM이 어디서 발생했는지 알기 어려움
- Error backtracking 이 이상한 곳으로 감 -> cuda 개념까지 알아봐야 함..
- 메모리의 이전 상황 파악이 어려움 <- OOM은 보통 iteration 돌면서 발생

### 해결 방법

- Batch Size 줄여서 GPU clean하고 Run
  - 코드를 제대로 짜면 해결됨
  - 코드 잘못 짜면 메모리가 쌓여서 이 방법도 안됨

참고 자료

- PyTorch에서 자주 발생하는 에러 질문들 : <https://pytorch.org/docs/stable/notes/faq.html>
- OOM시에 GPU 메모리 flush하기 : <https://pu-memory-after-a-runtimeerror/28781>
- GPU 에러 정리 : [https://brstar96.github.io/shoveling/device\\_error\\_summary/](https://brstar96.github.io/shoveling/device_error_summary/)

## ▼ GPUUtil

- nvidia-smi 처럼 GPU의 상태를 보여주는 모듈
  - nvidia-smi는 현재 시점의 스냅 샷만 보여주고 iteration마다 메모리 사용량 증가 확인 불가
- Colab 환경에서 GPU 상태 확인이 편함
- iteration마다 메모리 사용량 증가 여부 확인할 수 있음

```
!pip install GPUtil
```

```
Collecting GPUtil
  Downloading GPUtil-1.4.0.tar.gz (5.5 kB)
Building wheels for collected packages: GPUtil
  Building wheel for GPUtil (setup.py) ... done
  Created wheel for GPUtil: filename=GPUtil-1.4.0-py3-none-any.whl size=7411 sha256=7fc612a63
  Stored in directory: /root/.cache/pip/wheels/6e/f8/83/534c52482d6da64622ddbf72cd93c35d2ef28
Successfully built GPUtil
Installing collected packages: GPUtil
Successfully installed GPUtil-1.4.0
```

```
import GPUtil
```

```
GPUtil.showUtilization() # iter마다 하드웨어 사용량 확인 가능
```

```
| ID | GPU | MEM |
-----
| 0  | 0%  | 6%  |
```

## ▼ torch.cuda.empty\_cache()

- 사용되지 않은 GPU상 cache를 정리
- 가용 메모리를 확보
- del은 변수와 메모리주소의 관계를 풀어 메모리를 free
- reset,relunch 대신 쓰기 좋은 함수
- 학습 loop 시작 전에 사용하면 좋음

## ▼ 실습 코드

```
import torch
from GPUtil import showUtilization as gpu_usage
```



FROM OBJECT IMPORT OBJECTIZATION AS gpu\_daggo

```
tensorList = []
for x in range(10):
    tensorList.append(torch.randn(10000000, 10).cuda()) # 랜덤 값 cuda에 쌓기
```

```
# memory free하나 memory 공간 사용하려면 garbage collector 작동으로 캐시가 지워져야만 사용 가능
del tensorList
```

```
print("GPU Usage after deleting the Tensors")
gpu_usage()
```

```
print("GPU Usage after emptying the cache")
# 메모리 확보 강제
torch.cuda.empty_cache()
gpu_usage()
```

```
Initial GPU Usage
| ID | GPU | MEM |
-----
| 0 | 0% | 6% |

GPU Usage after allocating a bunch of Tensors
| ID | GPU | MEM |
-----
| 0 | 20% | 29% |

GPU Usage after deleting the Tensors
| ID | GPU | MEM |
-----
| 0 | 20% | 29% |

GPU Usage after emptying the cache
| ID | GPU | MEM |
-----
| 0 | 51% | 6% |
```

코드 출처 : <https://blog.paperspace.com/pytorch-memory-multi-gpu-debugging/>

- ▶ training loop에 tensor로 축적 되는 변수는 확인하자

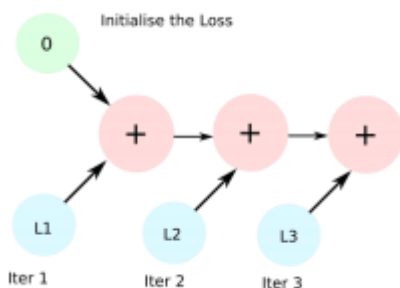
- tensor로 처리된 변수는 GPU 상에 메모리 사용
  - required\_grad=True면 grad하기 위해 메모리 버퍼도 사용
- 해당 변수(tensor로 처리된 변수)가 loop 안에 연산 있을 때 GPU에 computational graph를 생성(메모리 잠식)

### ▼ 실습 코드

```
total_loss = 0
for i in range(10000):
    optimizer.zero_grad()
    output = model(input)
    loss = criterion(output) # tensor
    loss.backward() # grad 존재
    optimizer.step()
    total_loss += loss # loss가 계속 쌓임
```

- 1-d tensor의 경우 python 기본 객체로 변환하여 처리할 것

코드 출처 : <https://blog.paperspace.com/pytorch-memory-multi-gpu-debugging/>



```
total_loss = 0
for x in range(10):
    # assume loss is computed
    iter_loss = torch.randn(3,4).mean()
    iter_loss.requires_grad = True

    # 방법 1
    total_loss += iter_loss.item

    # 방법 2
    # total_loss += float(iter_loss)
```

## ▼ del 명령어 사용

- 필요가 없어진 변수는 삭제
- python의 메모리 배치 특성 때문에 loop 이 끝나도 메모리를 점유

## ✔ 실습 코드

## ▼ batch 사이즈 실험

- 학습 시 OOM 이 발생했다면 batch 사이즈를 1로 실험

### ▶ 실습 코드

↳ 숨겨진 셀 1개

## ▼ torch.no\_grad() 사용

- Inference 할 때 torch.no\_grad() 구문을 사용
- backward pass 로 인해 쌓이는 메모리 사용량을 줄일 수 있음

### ▶ 실습 코드

↳ 숨겨진 셀 1개

## 예상치 못한 에러 메시지

- OOM 외에도 유사한 에러들이 발생
  - CUDNN\_STATUS\_NOT\_INIT : GPU 드라이버 문제
  - device-side-assert : OOM의 일종
- 위 에러도 cuda와 관련하여 OOM의 일종으로 간주되며 적절한 코드 처리의 필요함
- 해결 방법 참고자료 : [https://brstar96.github.io/shoveling/device\\_error\\_summary/](https://brstar96.github.io/shoveling/device_error_summary/)

## 기타 오류 해결법

- colab에서 너무 큰 사이즈의 모델 만들고 실행하지 말 것
  - 예: LSTM은 메모리를 많이 사용한다.
- CNN은 대부분 크기가 안 맞아서 오류가 발생함
  - torchsummary 등으로 사이즈를 맞춰야 함
  - CNN 끝단에서 linear로 변경
- tensor의 float precision을 16bit로 줄임
  - 잘 쓰지 않음, 최후의 수단으로 고려