

Deep Learning Framework Lab

Sehyun Park

Fall, 2023

E-mail: ps_hyen@snu.ac.kr

Outline

1. Preliminaries
2. Why do we need DL frameworks?
3. DL frameworks
4. Training a neural network

1. Preliminaries


To make your own model, you need...

- Hardware: Computer
- Software: Computing environment

To make your own model, you need. . .

- Hardware: (Fast) Computer (possibly with fast GPU)
- Software: (Python) Computing environment (with IDE and deep learning framework package such as Tensorflow or PyTorch, possibly with API for GPGPU)

Google Colab (<https://colab.research.google.com/>) provides an environment convenient for machine learning.



The screenshot shows the Google Colab web interface. The browser address bar displays <https://colab.research.google.com>. The page title is "Colaboratory에 오신 것을 환영합니다" (Welcome to Colaboratory). The interface includes a left sidebar with navigation options like "시작하기" (Get started), "데이터 과학" (Data science), and "추천 예시" (Recommended examples). The main workspace is titled "시작하기" (Get started) and contains the following text:

지금 읽고 계신 문서는 정적 웹페이지가 아니라 코드를 작성하고 실행할 수 있는 대화형 환경인 Colab 메모장입니다. 예를 들어 다음을 값을 계산하여 변수로 저장하고 결과를 출력하는 간단한 Python 스크립트가 포함된 코드 셀입니다.

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

06400

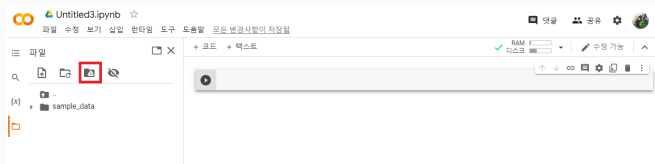
위의 코드를 실행하려면 셀을 클릭하여 선택한 후 코드 왼쪽의 실행 버튼을 누르거나 단축키 'Command/Ctrl+Enter'를 사용하세요. 셀을 클릭하면 코드 수정을 바로 시작할 수 있습니다. 특정 셀에서 정의한 변수를 나중에 다른 셀에서 사용할 수 있습니다.

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

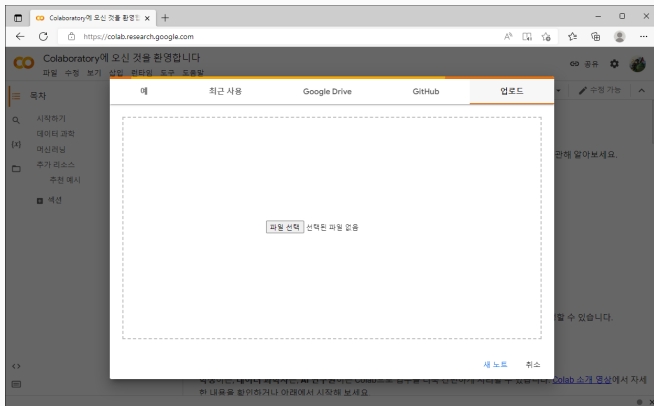
604800

Colab 메모장을 사용하면 실행 코드와 서식 있는 텍스트를 이미지, HTML, LaTeX 등과 함께 하나의 문서로 통합할 수 있습니다. Colab 메모장을 만들면 Google Drive 계정에 저장됩니다. Colab 메모장을 간편하게 공유하여 동료나 친구들이 댓글을 달거나 수정하도록 할 수 있습니다. 자세한 알아보려면 [Colab 개요](#)를 참조하세요. 새 Colab 메모장을 만들려면 위의 파일 메뉴를 사용하거나 다음 링크로 이동하세요.

- Colab Session is **temporary**.
 - Mount your Google Drive to preserve codes and data.



Upload all .ipynb and data files.



You should be able to handle basic Python:

1. Data types
2. Basic control flow, Functions and Classes
3. Modules and Packages (Numpy, Matplotlib, Pandas, Scikit-Learn)

2. Why do we need DL frameworks?

How do we make deep learning model?

- “Model training” can be understood as finding

$$\theta^* := \operatorname{argmin}_{\theta} \mathbb{E}_{\mathbf{X} \sim \mathcal{P}_{\mathbf{X}}} \ell(f(\mathbf{X}; \theta)),$$

where $f(x; \theta) = f_L^{\theta_L} \circ \dots \circ f_2^{\theta_2} \circ f_1^{\theta_1}(x)$.

- **Stochastic gradient descent (SGD)** update

$$\theta^k = \theta^{k-1} - \gamma_k \sum_{i=1}^N \nabla_{\theta} \ell(f(X_i; \theta^{k-1})), \quad X_i \stackrel{\text{i.i.d}}{\sim} \mathcal{P}_{\mathbf{X}}$$

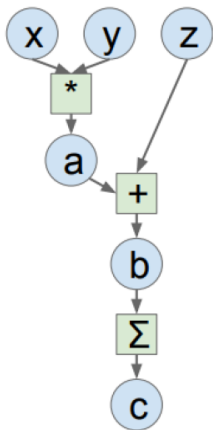
can estimate θ^* for proper γ_k , f , and ℓ .

How do we get the gradients?

Backpropagation can derive the analytic gradient

$$\begin{aligned}\frac{\partial l}{\partial f_{L-1}} &= \frac{\partial l}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \\ \frac{\partial l}{\partial f_{L-2}} &= \frac{\partial l}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial f_{L-2}} \\ &\vdots \\ \frac{\partial l}{\partial f_1} &= \frac{\partial l}{\partial f_L} \cdots \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_1}\end{aligned}$$

Backpropagation example: Numpy



```
import numpy as np
np.random.seed(0)
```

```
N, D = 3, 4
```

```
x = np.random.randn(N, D)
```

```
y = np.random.randn(N, D)
```

```
z = np.random.randn(N, D)
```

```
a = x * y
```

```
b = a + z
```

```
c = np.sum(b)
```

```
grad_c = 1.0
```

```
grad_b = grad_c * np.ones((N,D))
```

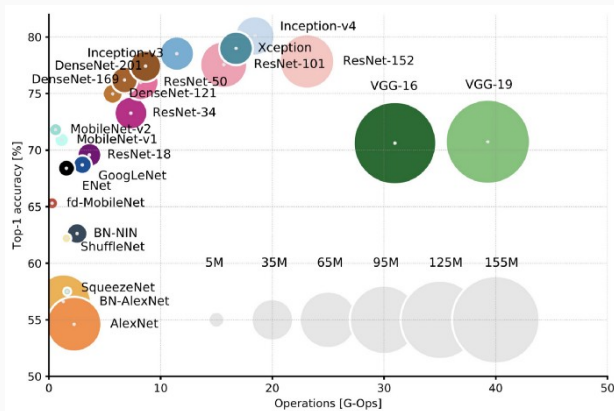
```
grad_a = grad_b.copy()
```

```
grad_z = grad_b.copy()
```

```
grad_x = grad_a * y
```

```
grad_y = grad_a * x
```

Can we make some models now?



Deep learning networks need lots of computations. . .

Ref. Canziani et al. "An Analysis of Deep Neural Network Models for Practical Applications."

Why do we need DL frameworks?

DL frameworks provide. . .

- Easy construction of computational graphs
- Automated gradient computing (by backpropagation) for common operations
- Acceleration with parallel computation (GPGPU)

GPU accelerates the training

Vertices:

{ (0.200, 0.000, 2.700),
(0.200, -0.112, 2.700),
... }

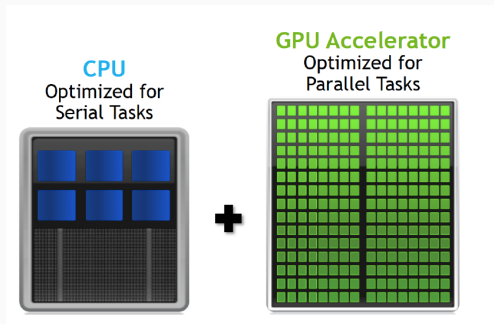


To render 3D images, we need to do:

- Projection of 3D coordinate into 2D surface
- Compute lighting and shading (on each pixel)
- Apply texture to the object (on each pixel)

⇒ Parallelizable operations

GPU accelerates the training



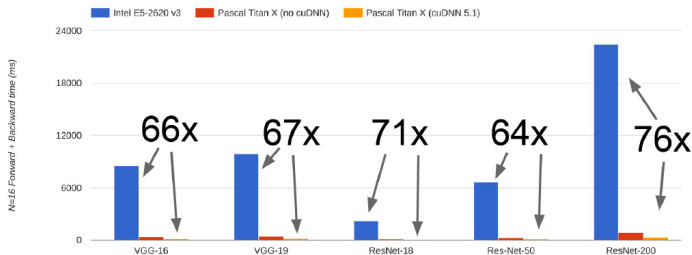
- GPU can also handle **General Purpose** computing.
- GPU has slower but more cores than CPU, which make parallel tasks faster.
- GPGPU relevant API: CUDA (for NVIDIA GPU), OpenCL

Ref. Nvidia

CPU vs GPU Benchmark

CPU vs GPU in practice

(CPU performance not well-optimized, a little unfair)



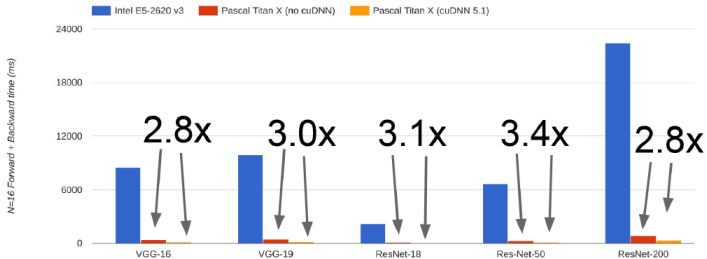
Data from <https://github.com/cjohnson/cnn-benchmarks>

Ref. Stanford cs231n

CPU vs GPU Benchmark

CPU vs GPU in practice

cuDNN much faster than
"unoptimized" CUDA



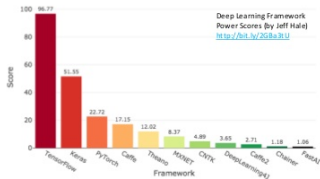
Data from <https://github.com/jcjohnson/cnn-benchmarks>

Ref. Stanford cs231n

3. DL frameworks

DL frameworks

Deep Learning Frameworks



Factors to consider:

- Learning curve
- Speed of development
- Size and passion of community
- Number of papers implemented in framework
- Likelihood of long-term growth and stability
- Ecosystem of tooling

1. TensorFlow
2. Keras
3. PyTorch
4. Caffe
5. theano
6. mxnet
7. CNTK
8. DL4J
9. Caffe2
10. Chainer
11. fast.ai



For the full list of references visit:
<https://ocw.mit.edu/deeplearning> [327]

<https://deeplearning.mit.edu> 2019

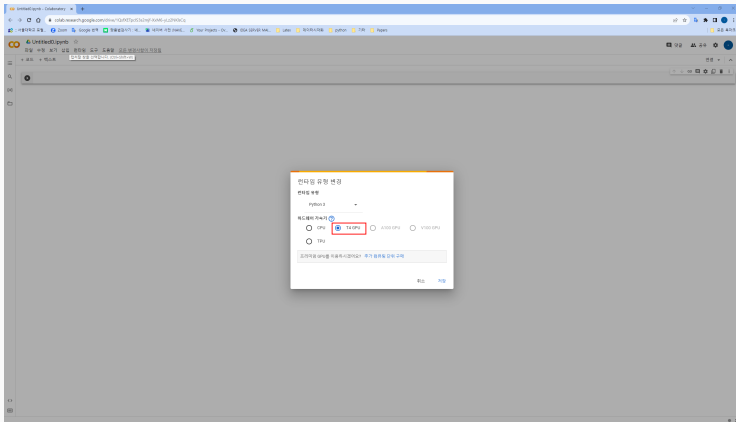
Source: <https://www.slideshare.net/noumfone/deep-learning-state-of-the-art-2019-mit-by-lex-fridman>

We will use **Tensorflow 2** and **PyTorch**.

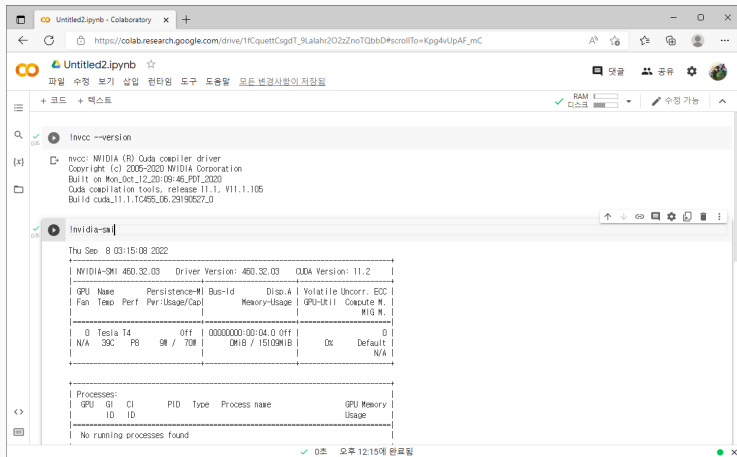
- In Google Colab, TF2, PyTorch is already installed.

Using GPU on Google Colab

RunTime -> Change Runtime type



Using GPU on Google Colab



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `https://colab.research.google.com/drive/1FCquettCsgdT_9Lalahr2Oz2zNoTQbbD#scrollTo=Kpg4vUpAF_mC`. The notebook title is "Untitled2.ipynb".

The notebook contains the following code cells:

```
Invoc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:09:46_PDT_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.TC465.06.29190527_0
```

```
nvdiags
```

The output of the `nvdiags` command is as follows:

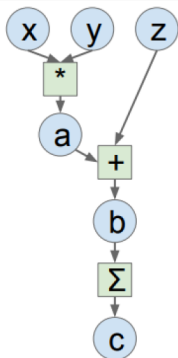
```
Thu Sep  8 03:15:08 2022
-----
| NVIDIA-SMI 460.32.03   Driver Version: 460.32.03   CUDA Version: 11.2   |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
| GPU Name      Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|  Memory-Usage | GPU-Util  Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
| 0   Tesla T4           Off   | 00000000:00:04:0 Off |                    0 |
| N/A   39C    P8      9W / 70W |  0MiB / 15109MiB |         0%   Default  |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
| Processes:                                                       GPU Memory |
|  GPU   GI    CI     PID    Type   Process name          Usage       |
|-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----|
| No running processes found
```

At the bottom of the notebook, the status bar indicates: `0초 오후 12:15에 완료됨`.

Tensorflow 2 and PyTorch use similar concepts.

- **Tensor**: A computation node. Similar with numpy array, but can be stored on GPU.
- **Automatic differentiation**: Usually denotes backpropagation with tensor.
 - Tensorflow 2 uses **GradientTape**
 - PyTorch stores gradient in tensor
- **Module**: A class representing a neural network; may store Tensors and learnable weights.

Backpropagation example: Tensorflow



```
import tensorflow as tf

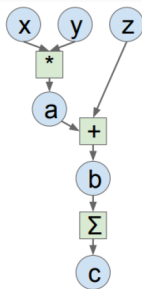
tf.random.set_seed(10)

N, D = 3, 4
x = tf.Variable(tf.random.normal((N, D)), name='x')
y = tf.Variable(tf.random.normal((N, D)), name='y')
z = tf.Variable(tf.random.normal((N, D)), name='z')

with tf.GradientTape() as tape:
    a = x * y
    b = a + z
    c = tf.reduce_sum(b)

grad_x, grad_y, grad_z = tape.gradient(c, [x,y,z])
```

Backpropagation example: PyTorch



```
import torch
torch.random.manual_seed(10)

device = torch.device('cuda:0') if torch.cuda.is_available() is True else None

N, D = 3, 4

x = torch.randn(N, D, requires_grad = True, device = device)
y = torch.randn(N, D, requires_grad = True, device = device)
z = torch.randn(N, D, requires_grad = True, device = device)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()

x.data, x.grad
```

More examples are in '4. Computational_graph and example' Notebook!

4. Training a neural network

5. simple_NN_with_TF_and_Pytorch.ipynb

- TensorFlow
 - Module can organize all attributes necessary for defining a neural network.
 - Using Keras, we can skip the implementation of a layer or a low-level training loop.
 - Keras also has useful utilities.
- Pytorch
 - `nn.Module` is similar with `tf.Module`.
 - `PyTorchLightning` module generate a training loop by defining a training loss.

Examples for 'MNIST' dataset with Keras and Pytorch.

- '6. keras_mnist_classification.ipynb'
- '7. torch_mnist_classification.ipynb'

Further considerations

- What if the layer we need is not implemented?
- How do we implement customized optimization algorithm?
- What if data are given in individual files (e.g. image)?

Read the tutorials, documentations, and codes of others!