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

Preliminaries

To make your own model, you need...

- Hardware: Computer
- Software: Computing environment

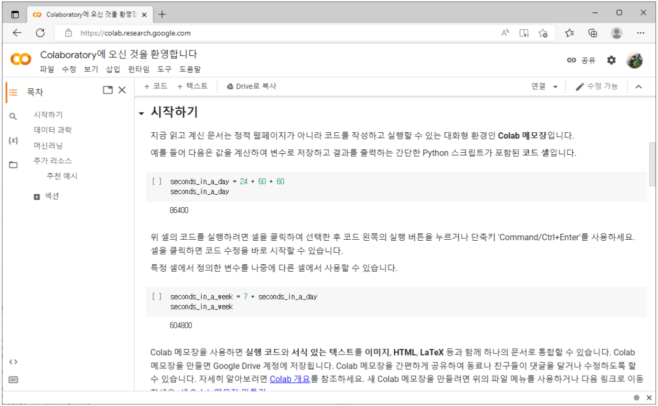
Preliminaries

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)

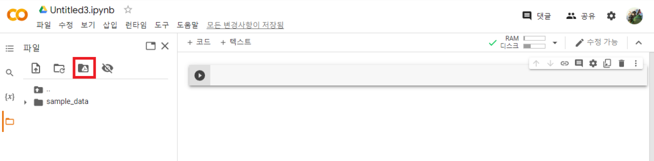
Preliminaries

Google Colab (https://colab.research.google.com/) provides an envrionment convinient for machine learning.



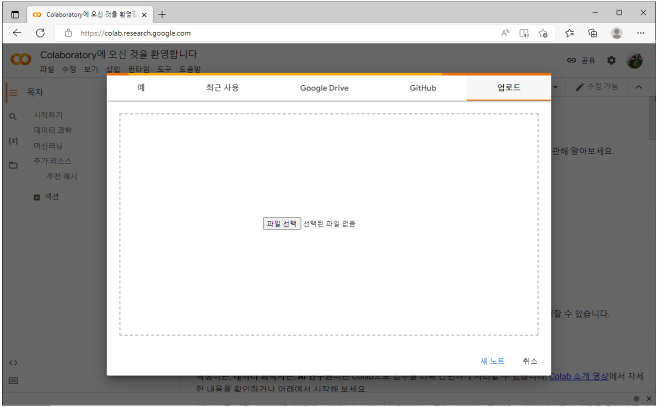
Preliminaries

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



Preliminaries

Upload all .ipynb and data files.



Preliminaries

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

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

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

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

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

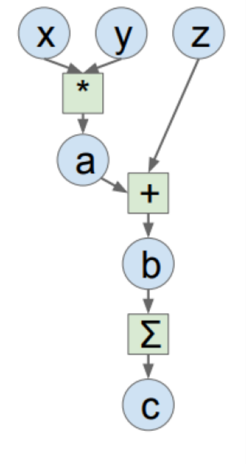
can estimate θ^* for proper γ_k, f, and ℓ.

How do we get the gradients?

Backpropagation can derive the analytic gradient

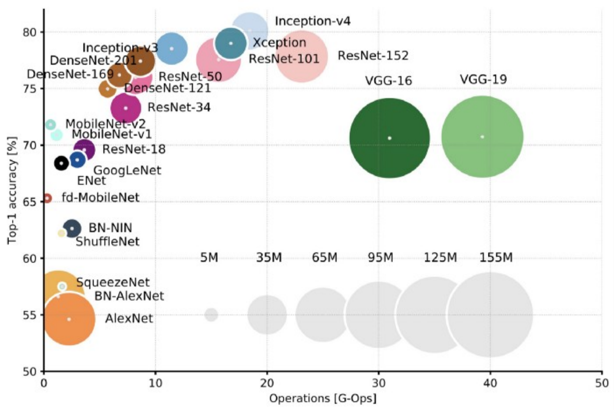
$$
\begin{aligned}
\frac{\partial \ell}{\partial f_{L-1}} &= \frac{\partial \ell}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \\
\frac{\partial \ell}{\partial f_{L-2}} &= \frac{\partial \ell}{\partial f_L} \cdot \frac{\partial f_L}{\partial f_{L-1}} \cdot \frac{\partial f_{L-1}}{\partial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

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. . .

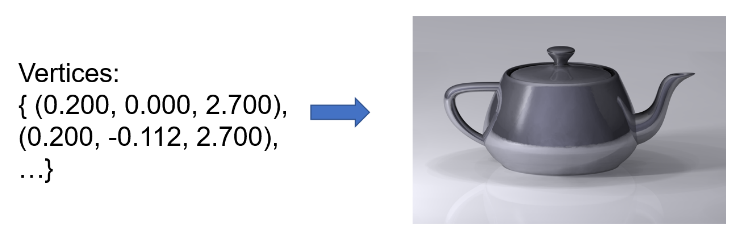
Ref. Canziani et al. "An Analysis of Deep Neural Network Models for Practical Applications."

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

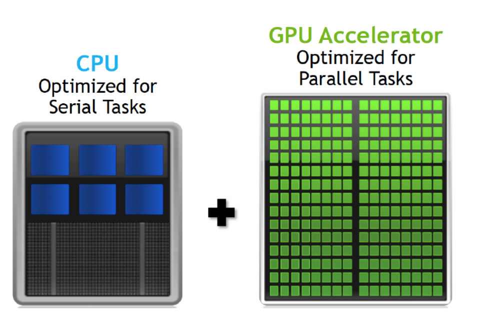


To render 3D images, we need to do:

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

$\Rightarrow$ Parallelizable operations

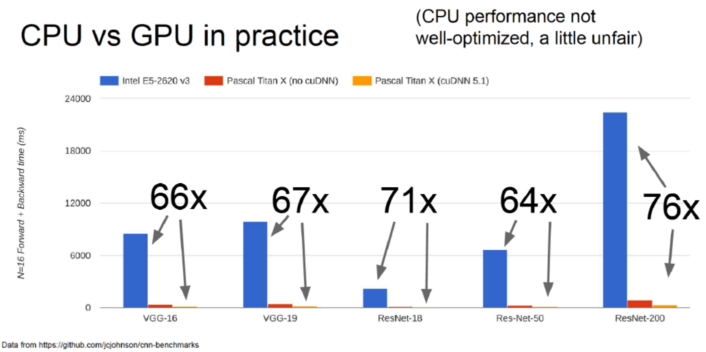
GPU accelerates the training



- GPU can also handle **G**eneral **P**urpose computing.
- GPU has slower but more cores than CPU, which make parallel tasks faster.
- GPGPU relevant API: CUDA (for NVIDA GPU), OpenCL

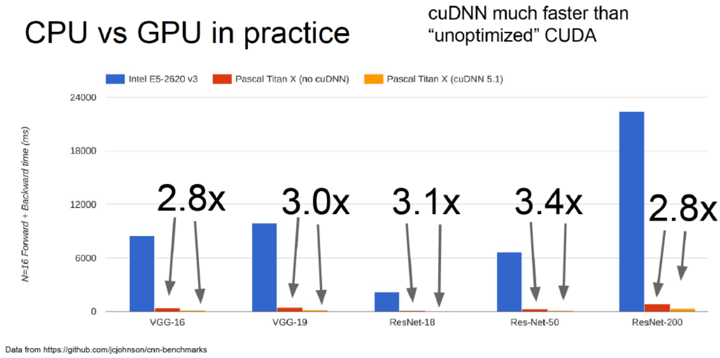
Ref. Nvidia

CPU vs GPU Benchmark



Ref. Stanford cs231n

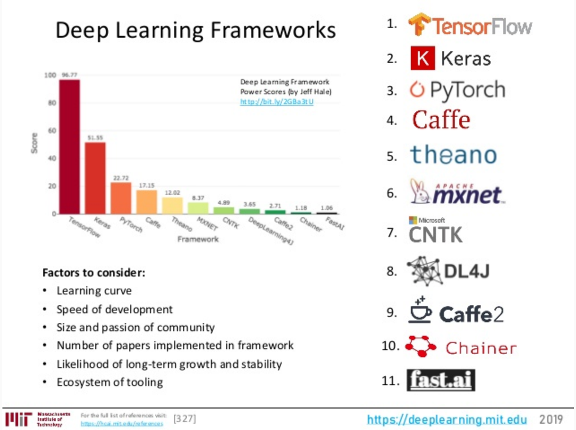
CPU vs GPU Benchmark



Ref. Stanford cs231n

3. DL frameworks

DL frameworks



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

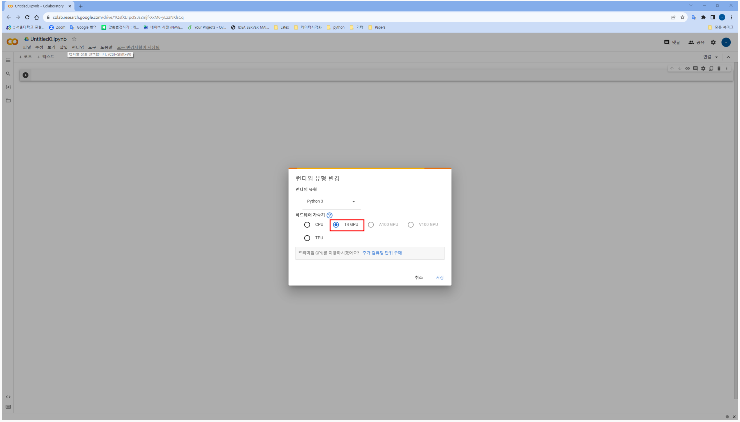
DL frameworks

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

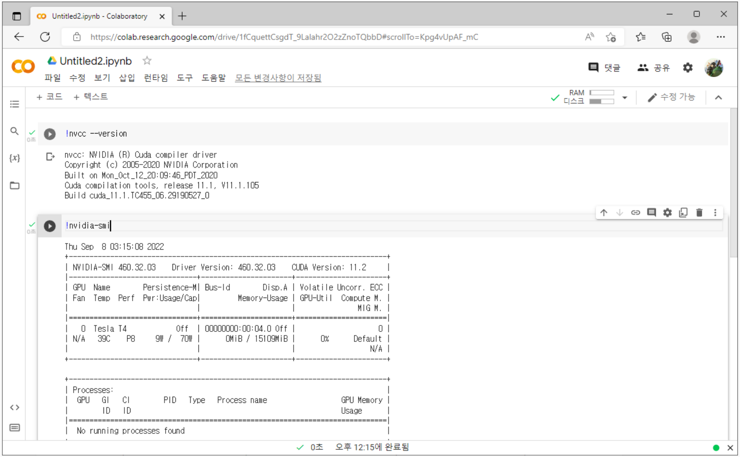
- In Google Colab, TF2, PyTorch is already installed.

Using GPU on Google Colab

RunTime -> Change Runtime type



Using GPU on Google Colab

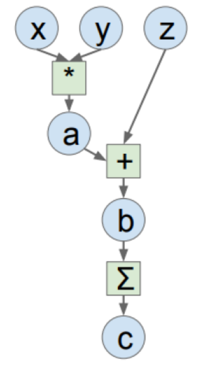


DL frameworks: Terminology

Tensorflow 2 and PyTorch use simliar concepts.

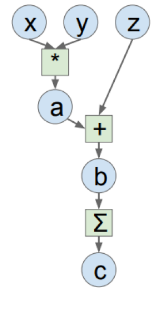
- **Tensor**: A coumputation node. Similar with numpy array, but can be stored on GPU.
- **Automatic differentiation**: Usually denotes backpropagation with tensor.
  - Tensorflow 2 uses **GradientTape**
  - PyTorch stores gradient in tensor
- **Module**: A class representing a neural network; may store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

TensorFlow and Pytorch

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.

Example

Examples for ‘MNIST’ dataset with Keras and Pytorch.

- ‘6. keras_mnist_classfication.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!