Basics of Machine Learning Tools

[CSED490X] Recent Trends in ML: A Large-Scale Perspective

Jungtaek Kim

jtkim@postech.ac.kr

POSTECH
Pohang 37673, Republic of Korea
https://jungtaek.github.io

March 16, 2022

Table of Contents

Large-Scale Datasets

Large-Scale Datasets

- The definition of large-scale datasets is inevitably unclear, but dataset cleaning and annotation for such datasets are challenging due to their size.
- Curated dataset:
  1. ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [Russakovsky et al., 2015];
  2. Open Images Dataset V6 (this link, partially);
  3. Wikipedia (in terms of specific categorization, e.g., topics);
  4. Conceptual 3M [Sharma et al., 2018] & Conceptual 12M [Changpinyo et al., 2021].
- Non-curated dataset (including datasets with machine-generated labels):
  1. JFT-300M [Sun et al., 2017] & JFT-3B [Zhai et al., 2021] (maybe);
  2. Open Images Dataset V6 (this link, mostly);
  3. Common Crawl (`https://commoncrawl.org`);
  4. ShapeNet [Chang et al., 2015] (mostly).

ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

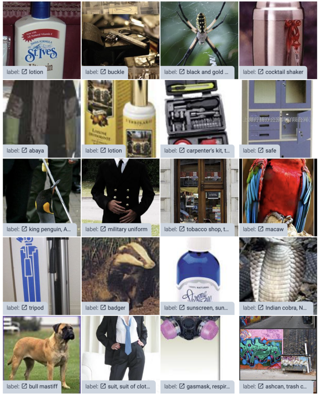


Figure 1: Examples of ILSVRC2012.

- It is to solve tasks for classification, classification with localization, and fine-grained classification.
- #Classes: 1,000
- #Training: 1,281,167
- #Validation: 50,000
- #Test: 100,000
- Details can be found in this link.

Figure 1 is taken from `https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=imagenet2012`.

Open Images Dataset V6

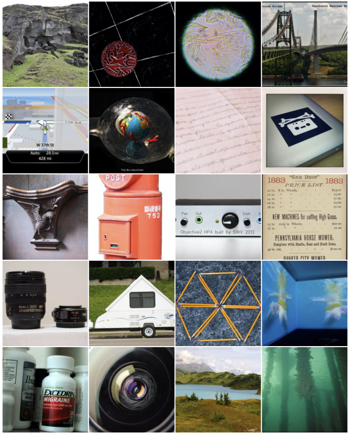


Figure 2: Examples of Open Images Dataset V4.

- It is a dataset of ~9M images annotated with image-level labels, object bounding boxes, object segmentation masks, visual relationships, and localized narratives.
- Details can be found in this link.

Figure 2 is taken from https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=open_images_v4.

Open Images Dataset V6

	Images	Labels	
		Machine-Generated	Human-Verified
#Training	9,011,219	164,819,642	57,524,352 (pos + neg)
#Validation	41,620	681,179	595,339 (pos + neg)
#Test	125,436	2,061,177	1,799,883 (pos + neg)
#Classes	–	15,387	19,957
#Trainable Classes	–	9,034	9,605

Conceptual 12M

- It is a dataset with $\sim$12 million image-text pairs meant to be used for vision-and-language pre-training.
- It covers a much more diverse set of visual concepts than the Conceptual 3M dataset [Sharma et al., 2018].
- Due to the proprietary rights, images are provided as image URLs.
- Details can be found in this link.

[Sharma et al., 2018] P. Sharma, N. Ding, S. Goodman, and R. Soricut. Conceptual Captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In Proceedings of the Annual Meeting of the Association for Computational Linguistics, pages 2556–2565, Melbourne, Australia, 2018.

Conceptual 12M

Figure 3: Examples of Conceptual 12M.

Taken from `https://github.com/google-research-datasets/conceptual-12m`.

Common Crawl

- It is a project that crawls the web and freely opens its archives.
- It generally crawls every month since 2008.
- This dataset is used in diverse language models.
- Details can be found in `https://commoncrawl.org`.

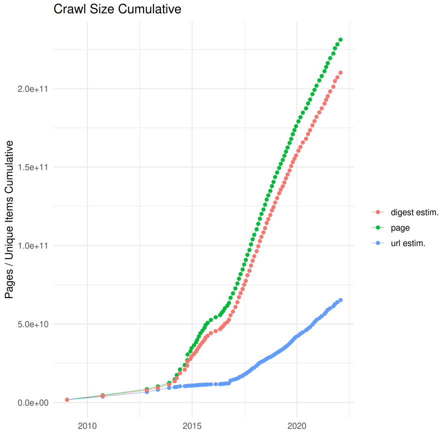


Figure 4: Cumulative size of Common Crawl.

Figure 4 is taken from `https://commoncrawl.github.io/cc-crawl-statistics/`.

Automatic Differentiation Frameworks

Automatic Differentiation Frameworks

- As discussed in the previous lectures, automatic differentiation is a key component of modern machine learning models.
- There are various projects for automatic differentiation: TensorFlow [Abadi et al., 2016], PyTorch [Paszke et al., 2019], Caffe, MXNet, and Theano.
- They support most of techniques in modern machine learning, e.g., a support for GPUs, parallelism, mixed-precision, diverse optimizers, and diverse layers.
- They are growing fast!

[Abadi et al., 2016] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 265–283, Savannah, Georgia, USA, 2016.

[Paszke et al., 2019] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (NeurIPS), volume 32, Vancouver, British Columbia, Canada, 2019.

TensorFlow vs. PyTorch

Taken from Google Trends.

Sample Code with PyTorch

```python
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Taken from this link.

Sample Code with PyTorch

Import packages

```
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Taken from this link.

Sample Code with PyTorch

Prepare for a dataset

```
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Taken from this link.

Sample Code with PyTorch

Create a model

```
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Taken from this link.

Sample Code with PyTorch

Declare a loss and an optimizer

```
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Taken from this link.

Sample Code with PyTorch

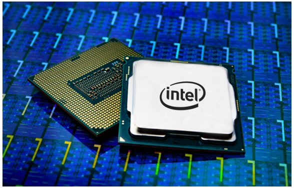
```python
import torch
import math

x = torch.linspace(-math.pi, math.pi, 2000)
y = torch.sin(x)

p = torch.tensor([1, 2, 3])
xx = x.unsqueeze(-1).pow(p)

model = torch.nn.Sequential(
    torch.nn.Linear(3, 1),
    torch.nn.Flatten(0, 1)
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-3
optimizer = torch.optim.RMSprop(model.parameters(),
                                lr=learning_rate)
for t in range(2000):
    y_pred = model(xx)

    loss = loss_fn(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

Train the model

Taken from this link.

Machine Learning Accelerators

Machine Learning Accelerators

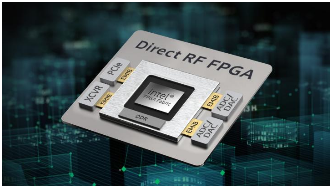


(a) CPU

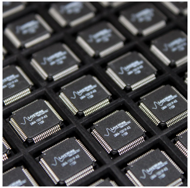
NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			
GPU Memory	40GB HBM2	80GB HBM2e	40GB HBM2	80GB HBM2e
GPU Memory Bandwidth	1,555GB/s	1,935GB/s	1,555GB/s	2,039GB/s
Max Thermal Design Power (TDP)	250W	300W	400W	400W

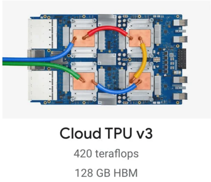
(b) GPU



(c) FPGA



(d) ASIC



(e) TPU

Figure 5: Examples of machine learning accelerators.

Figure 5(a) is taken from this link; Figure 5(b) is taken from this link; Figure 5(c) is taken from this link; Figure 5(d) is taken from Wikipedia; Figure 5(e) is taken from this link.

Machine Learning Accelerators

- Model training and inference are accomplished by running them on a machine learning accelerator.
- There are diverse types of accelerators:
  1. a central processing unit (CPU);
  2. a graphics processing unit (GPU);
  3. a field-programmable gate array (FPGA);
  4. an application-specific integrated circuit (ASIC);
  5. a tensor processing unit (TPU).
- Because of relatively cheap price and a relatively large number of threads, GPUs are dominant now.

Why Are GPUs More Popular Than CPUs in Machine Learning?

- GPUs are specialized in computing the same operations over a set of input data simultaneously.
- In particular, in the perspective of large-scale machine learning, GPUs are specialized in linear algebra operations such as matrix-vector multiplication and matrix-matrix multiplication.
- CUDA, developed by NVIDIA, is widely used in this field, whereas other tech companies such as Intel and AMD struggle to spread their own programs.
- Therefore, unfortunately, NVIDIA graphics cards are the only option we have.

Why Are GPUs More Popular Than CPUs in Machine Learning?

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Taken from this link.

How About Convolution Operations?

- Suppose that a data $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$ and a kernel $\mathbf{w} = [w_1, w_2, w_3]$ are given.
- The result of 1D convolution with zero padding is

$$\begin{bmatrix} w_3 x_1 \\ w_2 x_1 + w_3 x_2 \\ w_1 x_1 + w_2 x_2 + w_3 x_3 \\ w_1 x_2 + w_2 x_3 + w_3 x_4 \\ w_1 x_3 + w_2 x_4 + w_3 x_5 \\ w_1 x_4 + w_2 x_5 \\ w_1 x_5 \end{bmatrix}. \tag{1}$$

- By converting $\mathbf{x}$ to the Toeplitz matrix, (1) can be expressed as

$$\begin{bmatrix} 0 & 0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ 0 & x_1 & x_2 & x_3 & x_4 & x_5 & 0 \\ x_1 & x_2 & x_3 & x_4 & x_5 & 0 & 0 \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}. \tag{2}$$

Floating Point Operations (FLOPs)

- It is a measure of the amount of computations.
- Unlike FLOPs, floating point operations per second (FLOPS) is a measure of computer performance.
- FLOPs is widely used in comparing machine learning models where respective models are completely different.
- It is usually computed by extra software, e.g., `https://github.com/facebookresearch/fvcore`.

Cost of Computing

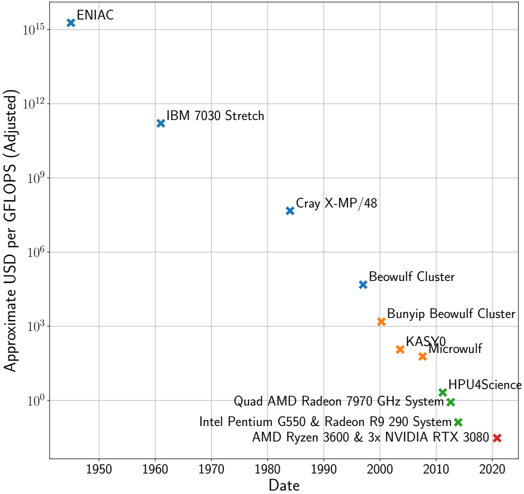


Figure 6: Approximate USD per GFLOPS vs. Date. Costs are adjusted based on the cost in 2020.

Floating-Point Arithmetic

- In computer systems, all real numbers are expressed by binary floating-point numbers:
$$\text{fraction} \times \text{base}^{\text{exponent}}, \tag{3}$$
where $\text{base} = 2$.

- For example, a decimal real number 123.456 is
$$1.11101101110100101111001 \times 2^{6}, \tag{4}$$
as a binary single-precision number.

- According to the IEEE 754 standard, it is stored as
$$0 \mid 10000101 \mid 11101101110100101111001. \tag{5}$$

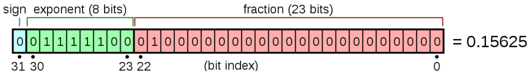


Figure 7: Example of a layout for 32-bit floating point.

Figure 7 is taken from Wikipedia.

Floating-Point Arithmetic

- Low-precision arithmetic, e.g., 16-bit floating point (a.k.a. half-precision), is beneficial for
  1. running more operations;
  2. reducing memory usage;
  3. reducing communication costs;
  4. using less energy,

  in machine learning.

- Many machine learning accelerators support low-precision arithmetic.

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			
GPU Memory	40GB HBM2	80GB HBM2e	40GB HBM2	80GB HBM2e
GPU Memory Bandwidth	1,555GB/s	1,935GB/s	1,555GB/s	2,039GB/s
Max Thermal Design Power (TDP)	250W	300W	400W	400W

Figure 8: Specifications of NVIDIA A100.

Figure 8 is taken from this link.

Any Questions?

References I

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, Savannah, Georgia, USA, 2016.

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015.

S. Changpinyo, P. Sharma, N. Ding, and R. Soricut. Conceptual 12M: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Virtual, 2021.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, Vancouver, British Columbia, Canada, 2019.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

P. Sharma, N. Ding, S. Goodman, and R. Soricut. Conceptual Captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 2556–2565, Melbourne, Australia, 2018.

C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 843–852, Venice, Italy, 2017.

X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. Scaling vision transformers. *arXiv preprint arXiv:2106.04560*, 2021.