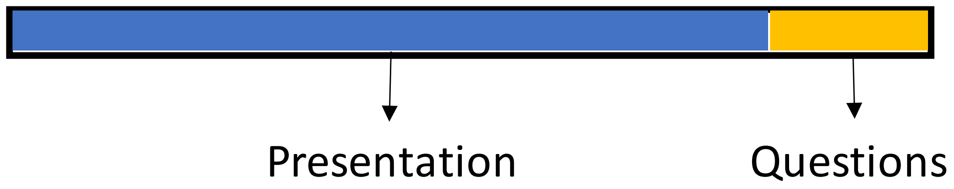
Machine Learning Tools

Bazyli Polednia 2021

Plan for today

Total: 1h

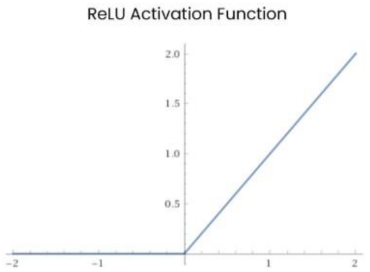


Deep Learning Boom in recent years

1. **Hardware and software**
   - Introduction of TPUs and more powerful GPUs
   - Parallelization of computations
   - Programming interfaces e.g. Nvidia CUDA
   - Deep Learning libraries and toolkits e.g. Tensorflow, PyTorch, Keras

2. **Datasets and benchmarks**
   - „Big Data”
   - Exponential growth of hardware storage
   - Public datasets e.g. ImageNet, Kaggle competitions

3. **Algorithmic advances**
   - New activation functions
   - New optimizers
   - Batch optimalization
   - …



Deep Learning Boom in recent years

1. **Hardware and software**
   - Introduction of TPUs and more powerful GPUs
   - Parallelization of computations
   - Programming interfaces e.g. Nvidia CUDA
   - Deep Learning libraries and toolkits e.g. Tensorflow, PyTorch, Keras

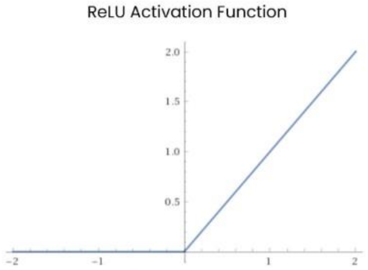
TensorFlow

2. **Datasets and benchmarks**
   - „Big Data"
   - Exponential growth of hardware storage
   - Public datasets e.g. ImageNet, Kaggle competitions

IMAGENET

kaggle

3. **Algorithmic advances**
   - New activation functions
   - New optimizers
   - Batch optimalization
   - …



CPU vs. GPU

CPU vs. GPU

CPU

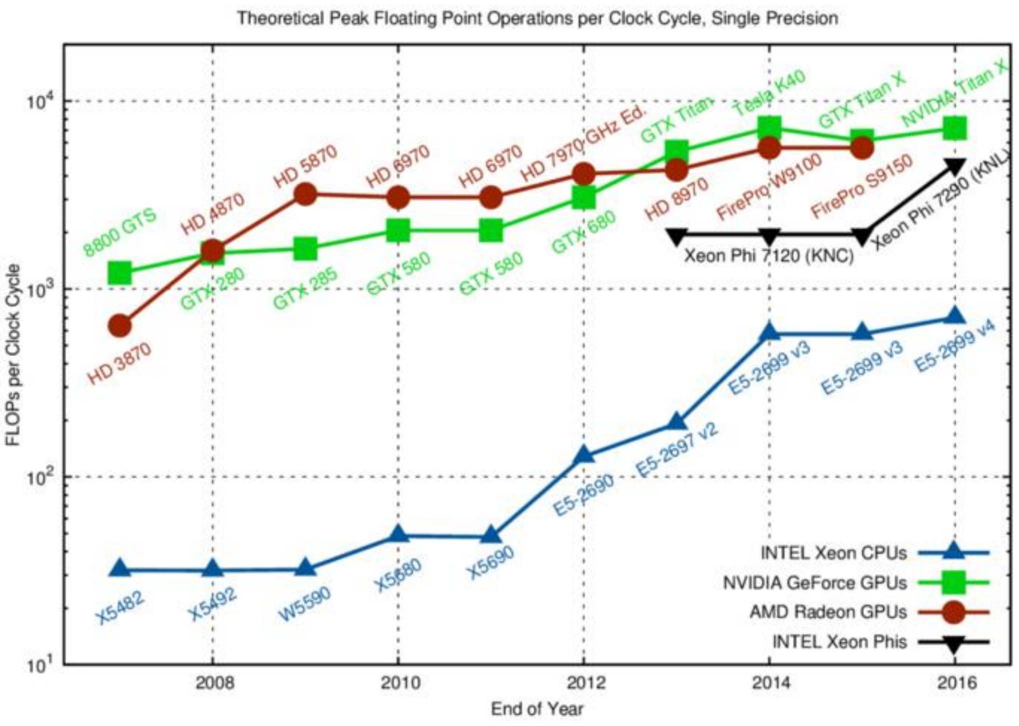
- Few complex cores
- Single-thread performance optimization
- Transistors dedicated to complex ILP
- Small die surface for integer and floating point units

GPU

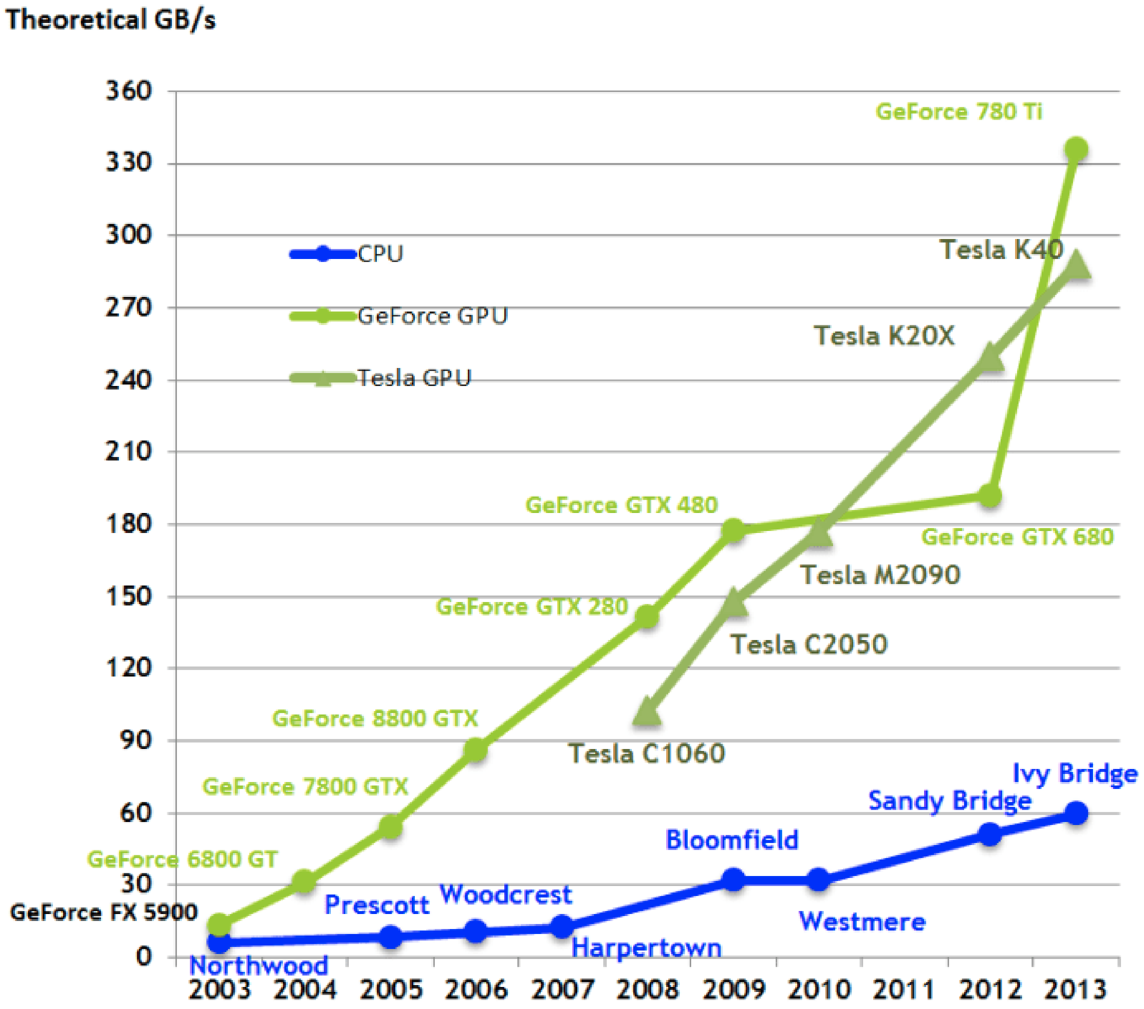
- Many simpler cores
- Many concurrent hardware threads
- Maximization of float point throughput
- Large die surface for integer and floating point units

GPUs are optimized for ***processing multiple computations*** simultaneously.

CPU vs. GPU

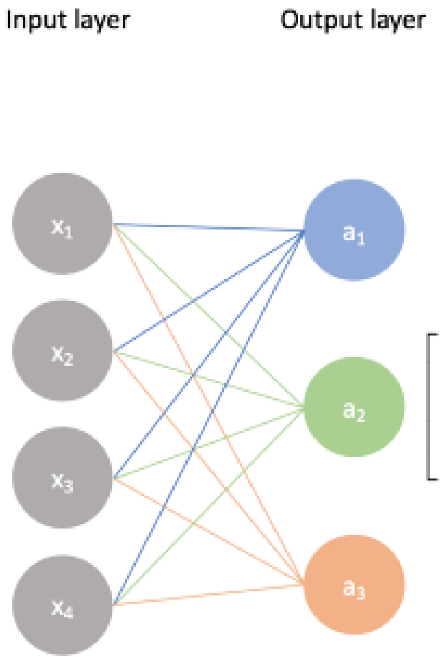


Theoretical Peak Floating Point Operations per Clock Cycle, Single Precision
FLOPs per Clock Cycle
End of Year
INTEL Xeon CPUs
NVIDIA GeForce GPUs
AMD Radeon GPUs
INTEL Xeon Phis
Xeon Phi 7120 (KNC)
Xeon Phi 7290 (KNL)



Theoretical GB/s
CPU
GeForce GPU
Tesla GPU
GeForce 780 Ti
Tesla K40
Tesla K20X
GeForce GTX 480
GeForce GTX 680
Tesla M2090
GeForce GTX 280
Tesla C2050
GeForce 8800 GTX
Tesla C1060
GeForce 7800 GTX
GeForce 6800 GT
GeForce FX 5900
Ivy Bridge
Sandy Bridge
Bloomfield
Prescott
Woodcrest
Westmere
Harpertown
Northwood

Why are GPUs used in DL?



A simple neural network

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[activation]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

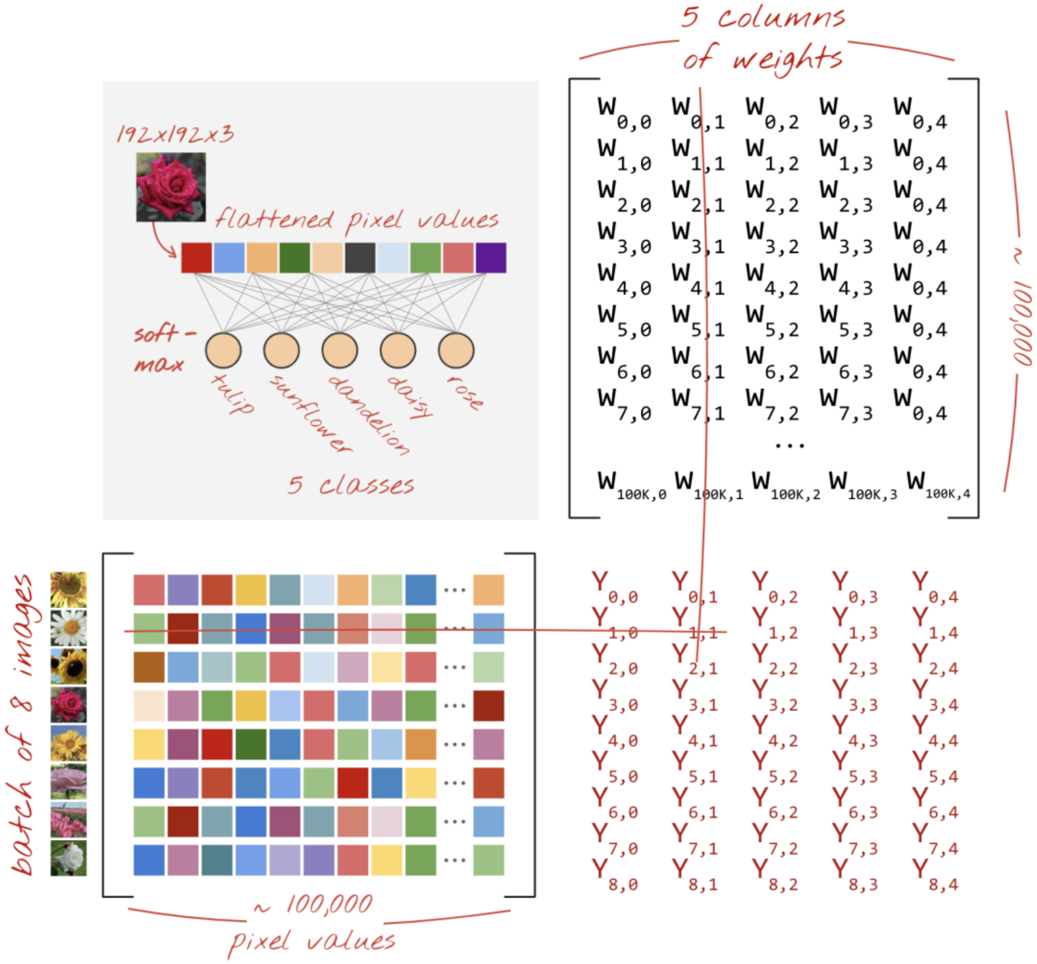
Can we do even better than GPU?

TPU – Tensor Processing Unit

Modern GPUs are organized around programmable "cores", a very flexible architecture that allows them to handle a variety of tasks such as 3D rendering, deep learning, physical simulations, etc.

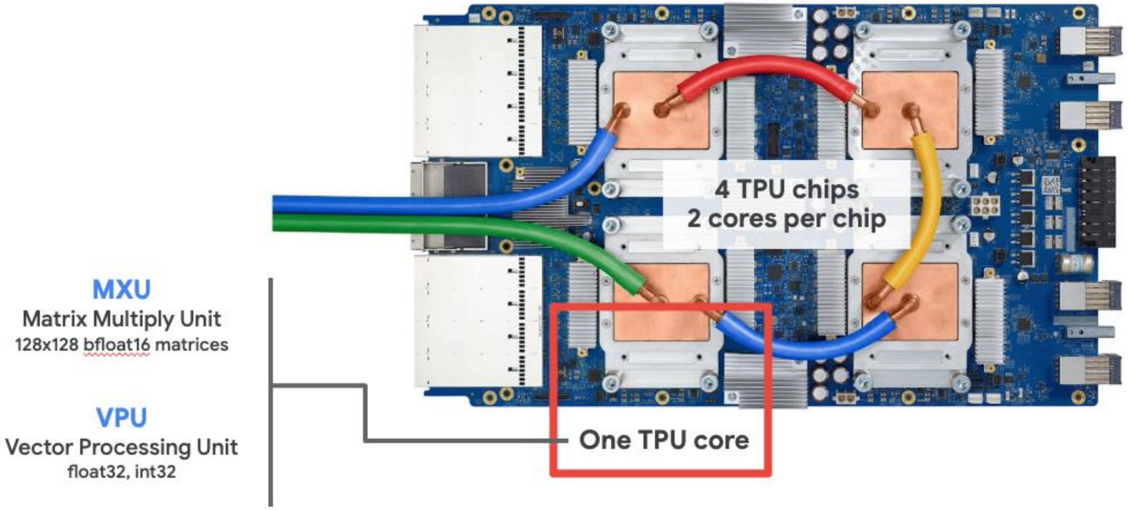
TPUs on the other hand pair a classic vector processor with a dedicated matrix multiply unit and excel at any task where large matrix multiplications dominate, such as neural networks.

TPU – Tensor Processing Unit

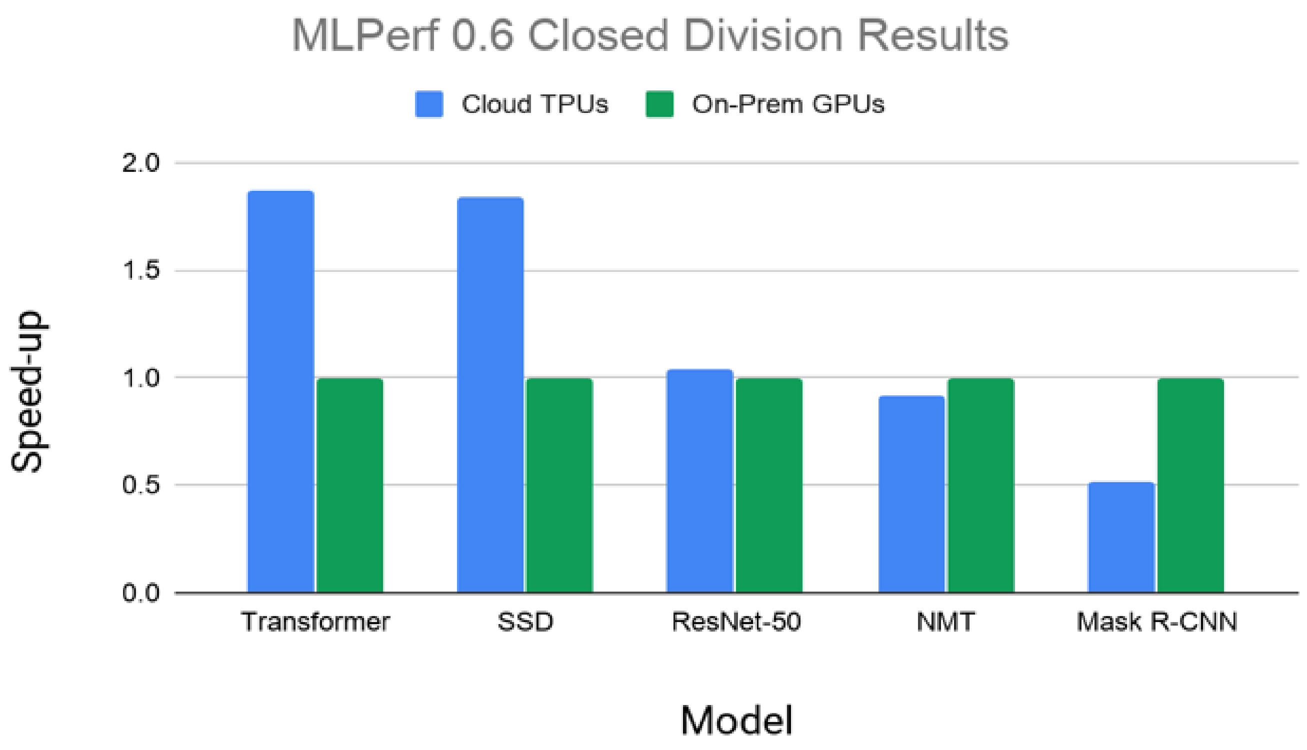


TPU – Tensor Processing Unit

MXU (Matrix Multiply Unit) – responsible for matrix multiplication
VPU (Vector Processing Unit) – handles other DL tasks such as activation functions

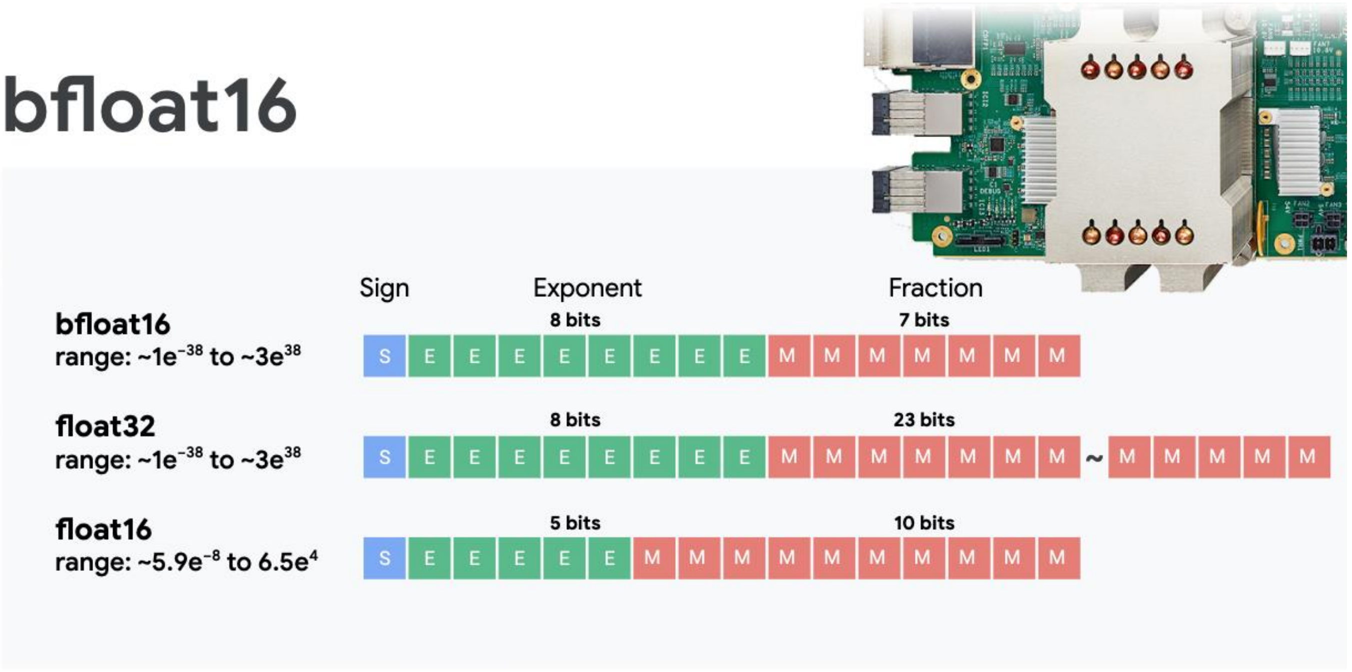


TPU – Tensor Processing Unit



Google Cloud TPU v3 Pod speed-ups over the largest-scale on-premise NVIDIA DGX-2h clusters entered in the MLPerf 0.6 Closed Division. The Cloud TPU Pod submissions use 1024, 1024, 1024, 512, and 128 chips respectively; the NVIDIA DGX-2h clusters use 480, 240, 1536, 256, and 192 chips respectively. [1,2]

bfloat16



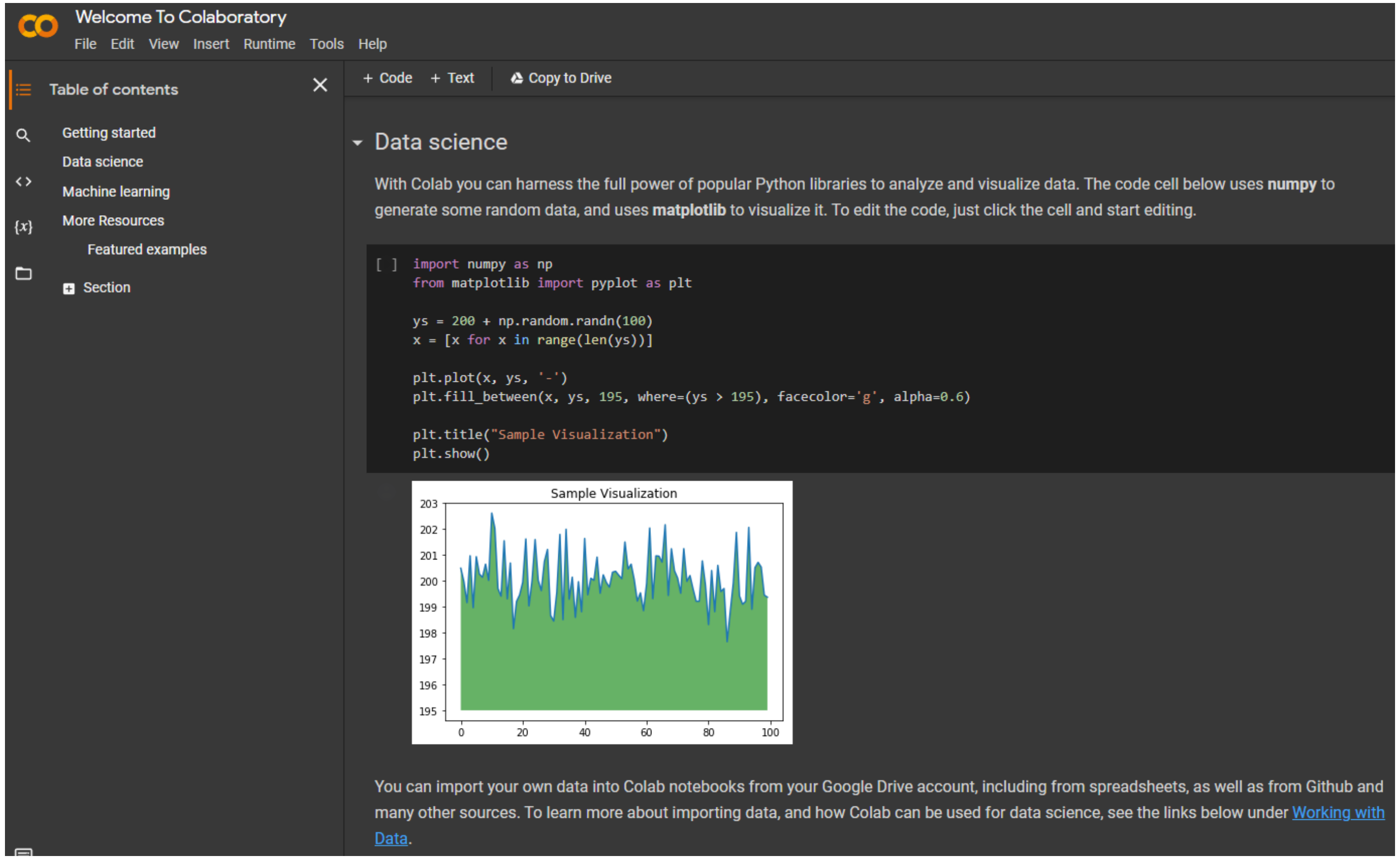
$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\dots b_{23})_2-127} \times (1.b_{22}b_{21}\dots b_0)_2.$$

Data in Deep Learning is already noisy due to sampling and measurement imprecision

Google Colab

Colab allows you to write
and execute Python in your browser, with:

- Zero configuration required
- Free access to GPUs and TPUs
- Easy sharing

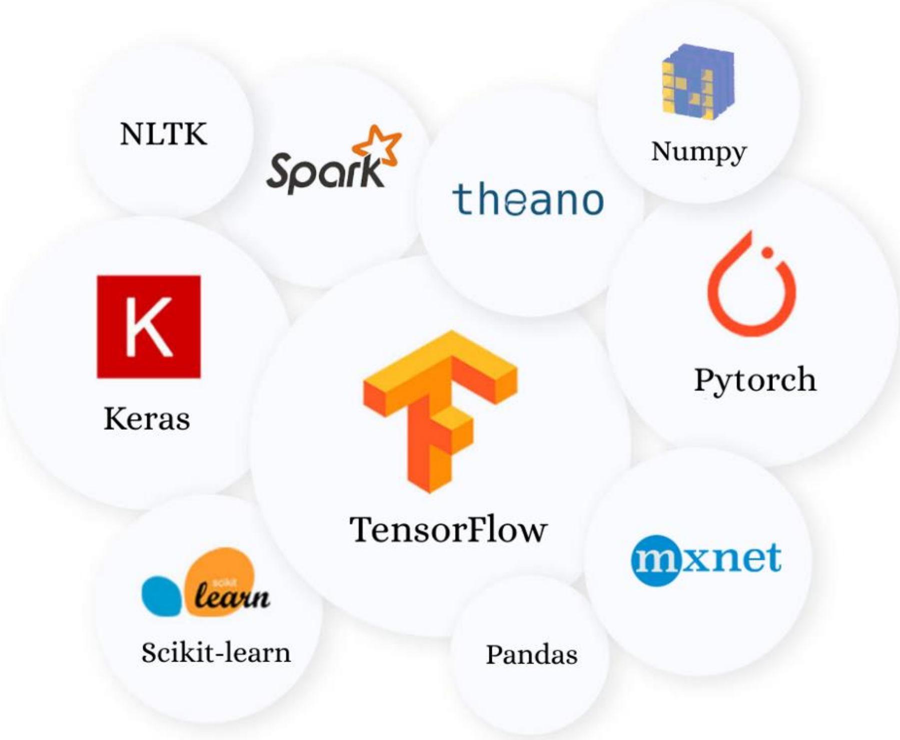


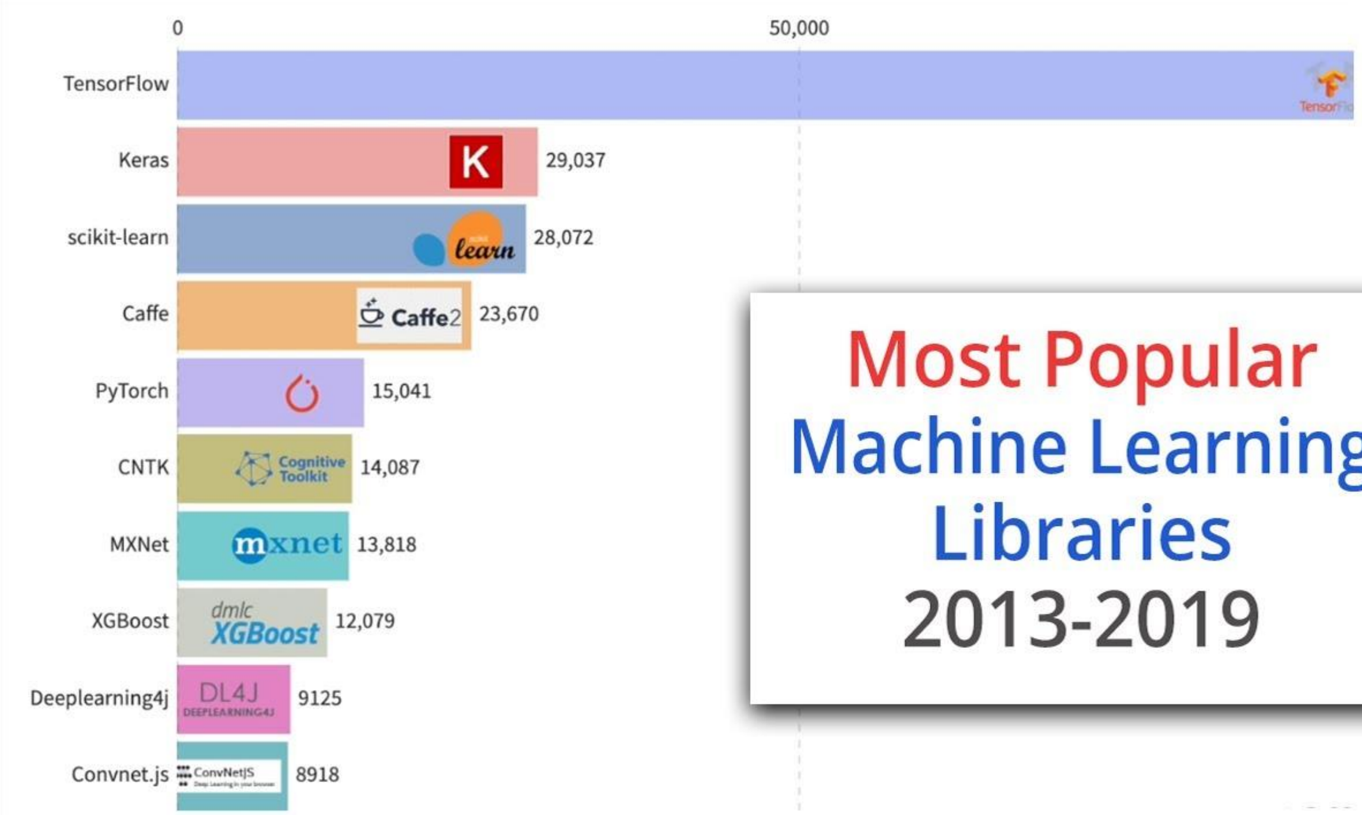
Hardware Accelarators

- Quick prototype and deployment
- Plug and play
- Support for OpenVINO toolkit
- Pre-trained models
- Model Optimizer
- Inference Engine

Hardware Accelarators

Machine Learning libraries and frameworks


NLTK
Spark
theano
Numpy
Keras
Pytorch
TensorFlow
mxnet
Scikit-learn
learn
Pandas



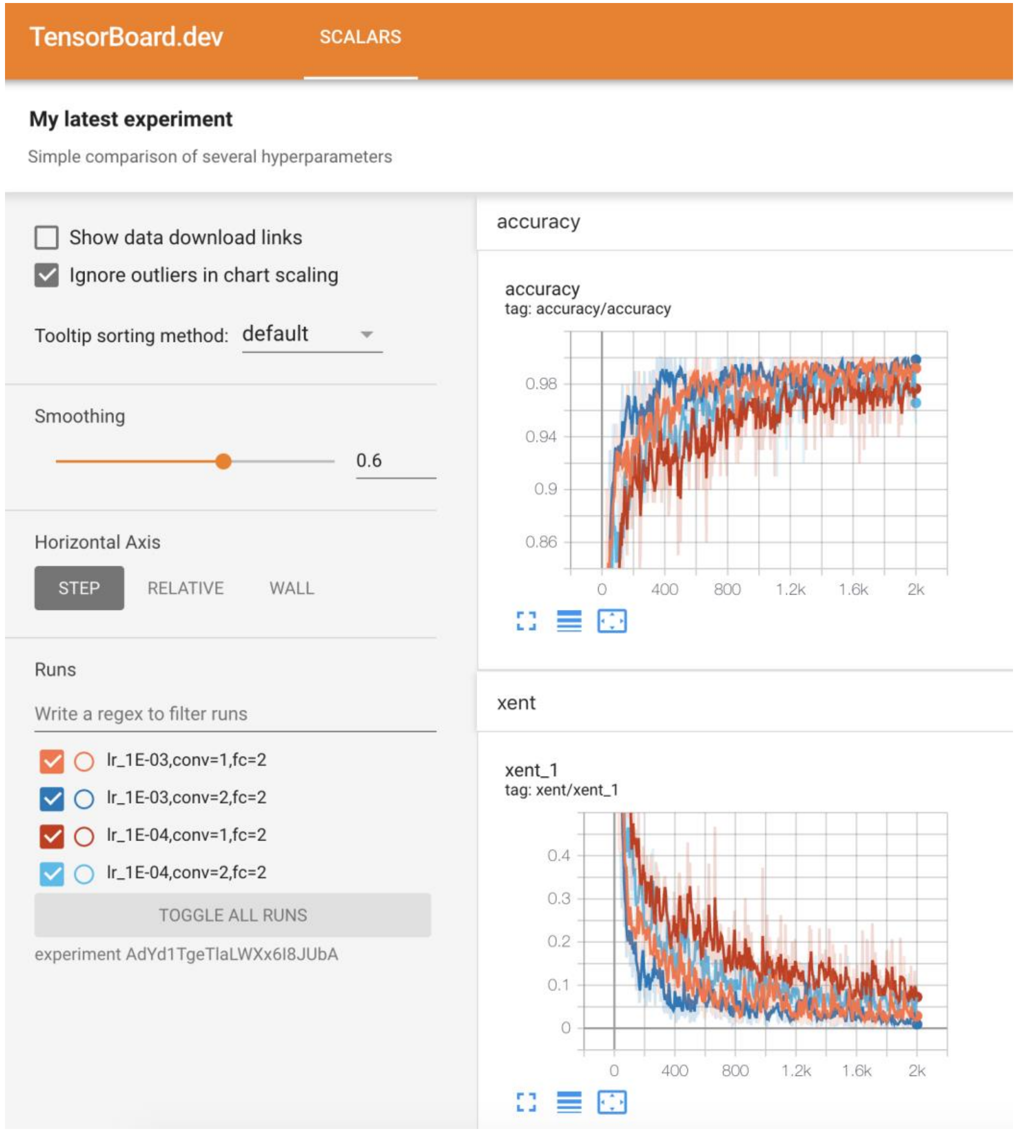
Most Popular Machine Learning Libraries 2013-2019

TensorFlow

- Developed by Google, released in 2015
- Offers level of abstraction while building DL models
- Support for multiple programming languages
- Offers cool visualization metrics

TensorFlow

TensorBoard



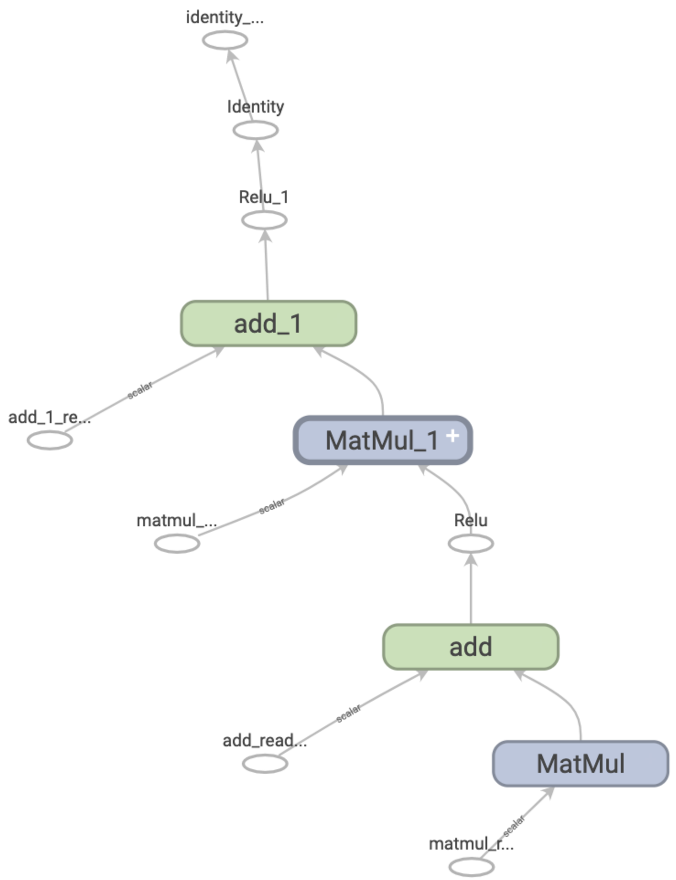
PyTorch

- Developed by Facebook, released in 2016
- Great support for distributed training and cloud computations
- Good documentation and online support community

Graph execution vs. eager execution

TensorFlow offers both eager execution and **graph execution** modes.

Graphs are data structures that contain a set of tf.Operation objects, which represent units of computation; and tf.Tensor objects, which represent the units of data that flow between operations.



Graphs are also easily optimized:

- Statically infer the value of tensors by folding constant nodes
- Separate sub-parts of a computation that are independent and split them between threads or devices.
- Simplify arithmetic operations by eliminating common subexpressions.

Automatic differentiation

Task: compute gradient for function $y=x^2$ at $x=3$

TensorFlow

```
x = tf.Variable(3.0)
with tf.GradientTape() as tape:
  y = x * x
dy_dx = tape.gradient(y, x)
```

PyTorch

```
x = torch.tensor([3.], requires_grad=True)
Q = x**2
external_grad = torch.tensor([1.])
Q.backward(gradient=external_grad)
x.grad
```

Simple neural network

Task: create a two-layer neural network

TensorFlow

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
model.compile(
    optimizer="adam",
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"],
)
```

PyTorch

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(128, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

net = Net()
```

Questions?

Thank you for attention