Pattern Analysis & Machine Intelligence Praktikum: MLPR-19

Martin Mundt, Dr. Iuliia Pliushch, Prof. Dr. Visvanathan Ramesh

Goethe Uni Frankfurt

Week 6: Introduction to ML/DL frameworks - PyTorch

DL software equirements

- N-dimensional matrices/tensors with common operations & convenient slicing
- Ability to group mathematical functions into more abstract building blocks (like layers, nets)
- Definition of complicated execution sequences of more than two building blocks (think of RNNs)
- A set of common training algorithms with parameter choices & sensible defaults
- Access to databases that are common in the machine learning domain

DL software requirements

- Compatibility with fundamental backends such as BLAS or GPU SDKs
- Seamless GPU & multi-GPU support (abstract away transfers & synchronization)
- Potential to distribute data & networks to multiple machines (or cloud)
- A "model-zoo", i.e. pre-trained models
- Target common Operating Systems
- Be installable & usable for non-experts
- Little computational & memory overhead through interfaces to e.g. Python

PyTorch

PyTorch

Tensor computation library
Express in Python (or LUA)
Underlying implementation: C, CUDA

PyTorch

Created by: Facebook AI
Initial beta release: January 2017
Latest stable release: PyTorch 1.1, April 2019 (1.0 autumn 18)
License: Open source - BSD-3

Addition to Torch version 7 (2012: same C backend + LUA & JIT, still maintained at http://torch.ch/)

Website: https://pytorch.org/
Git: https://github.com/pytorch/pytorch

PyTorch

KEY FEATURES & CAPABILITIES

See all Features >

Hybrid Front-End

A new hybrid front-end seamlessly transitions between eager mode and graph mode to provide both flexibility and speed.

Distributed Training

Scalable distributed training and performance optimization in research and production is enabled by the torch.distributed backend.

Python-First

Deep integration into Python allows popular libraries and packages to be used for easily writing neural network layers in Python.

Tools & Libraries

A rich ecosystem of tools and libraries extends PyTorch and supports development in computer vision, NLP and more.

https://pytorch.org/features

PyTorch

A summary of core features:

- **GPU-ready Tensor library**: if you use numpy, you have used Tensors.
- **Dynamic Neural Networks: Tape based Autograd** unique way of building neural networks: using and replaying a tape recorder.
- **Python first**
- **Auto-Differentiation**
- **Fast and Lean** At the core, CPU and GPU Tensor and Neural Network backends (TH, THC, THNN, THCUNN) are written as independent libraries with a C99 API. They are mature and have been tested for years.
- **Extensions without pain** You can write new neural network layers in Python using the torch API.
- **Torchvision** datasets and utility for computer vision.

PyTorch - tutorials

Before going on, let's look at the Getting Started PyTorch for Numpy users blitz and the autograd tutorial: https://pytorch.org/tutorials/index.html

PyTorch -Sequential example

```
import torch
matrix1 = torch.Tensor(3,3)
matrix2 = torch.Tensor(3,3)
product = torch.matmul(matrix1,matrix2)
print(product)
```

Neural Network

```
from torch import nn as nn
class Model(nn.Module):
        def __init__(self):
                super(Model, self).__init__()
                self.net = nn.Sequential(nn.Linear(2,2), nn.Linear(2,2))
        def forward(self, x):
                x = self.net(x)
                return x

model = Model()
result = model(torch.rand(2))
print(result)
```

PyTorch - (dynamically) defining forward

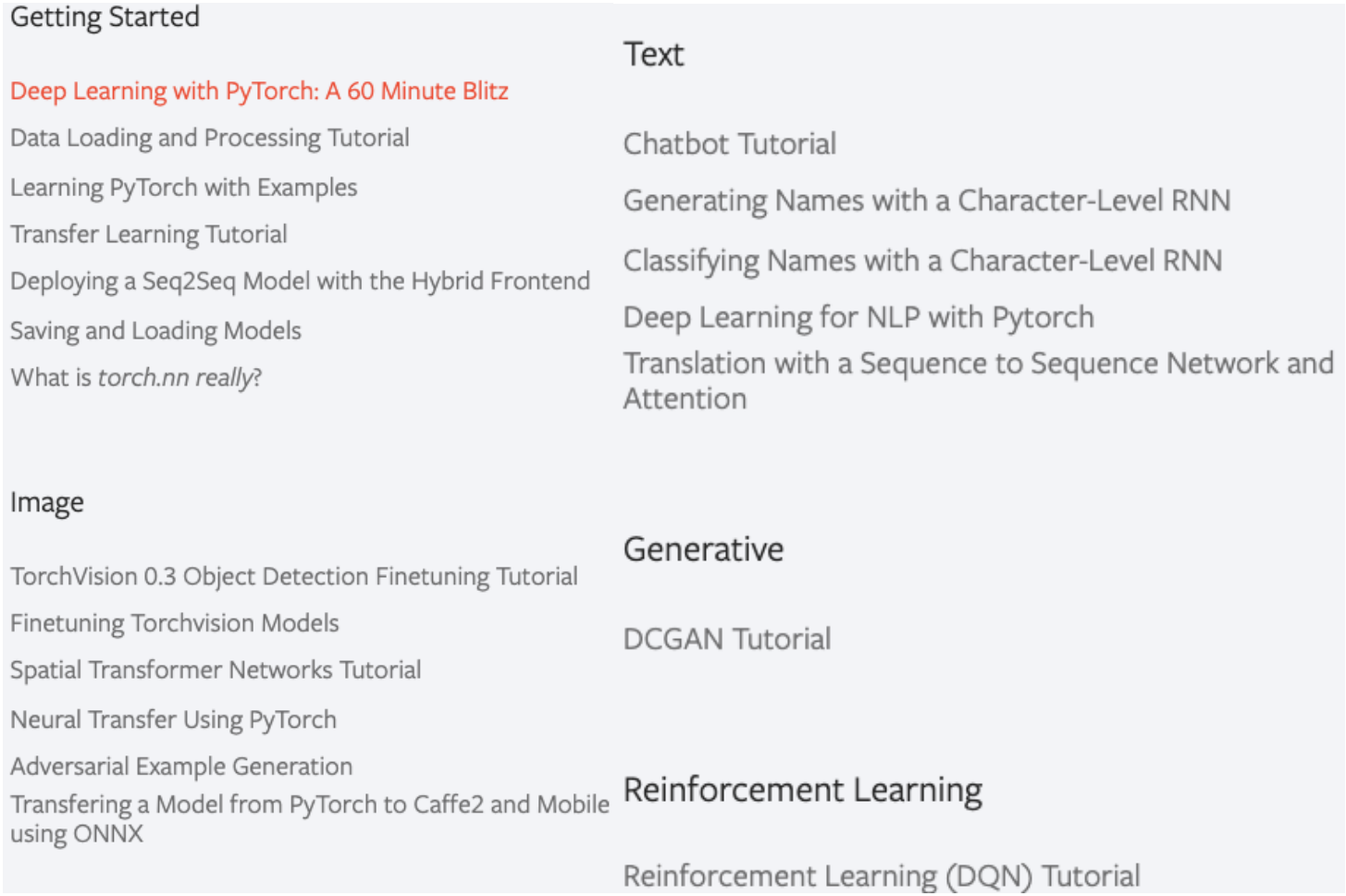
```
from torch import nn as nn
import torch.nn.functional as F

class Model(nn.Module):
        def __init__(self):
                super(Model, self).__init__()
                self.input_size = 28*28
                self.conv1 = nn.Conv2d(1, 32, 5)
                self.mp1 = nn.MaxPool2d(2, 2)
                self.conv2 = nn.Conv2d(32, 64, 5)
                self.mp2 = nn.MaxPool2d(2,2)
                self.fc = nn.Linear(64*4*4, 10)

        def forward(self, x):
                x = self.mp1(F.relu(self.conv1(x)))
        x = self.mp2(F.relu(self.conv2(x)))
                x = x.view(-1, 64*4*4)
                x = self.fc1(x)
                return x

model = Model()
result = model(torch.rand(1,1,28,28))
print(result)
```

PyTorch - Training the network with mini-batch SGD

```
epochs = 5
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
model = Model()

for e in range(epochs):
        for i, (inp, target) in enumerate(train_loader):
                output = model(inp)
                loss = criterion(output, target)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

        # cross-validation, testing etc.
```

PyTorch - tutorials

- Large community, many official tutorials, blogs and GitHub.



https://pytorch.org/tutorials/index.html

PyTorch - internals

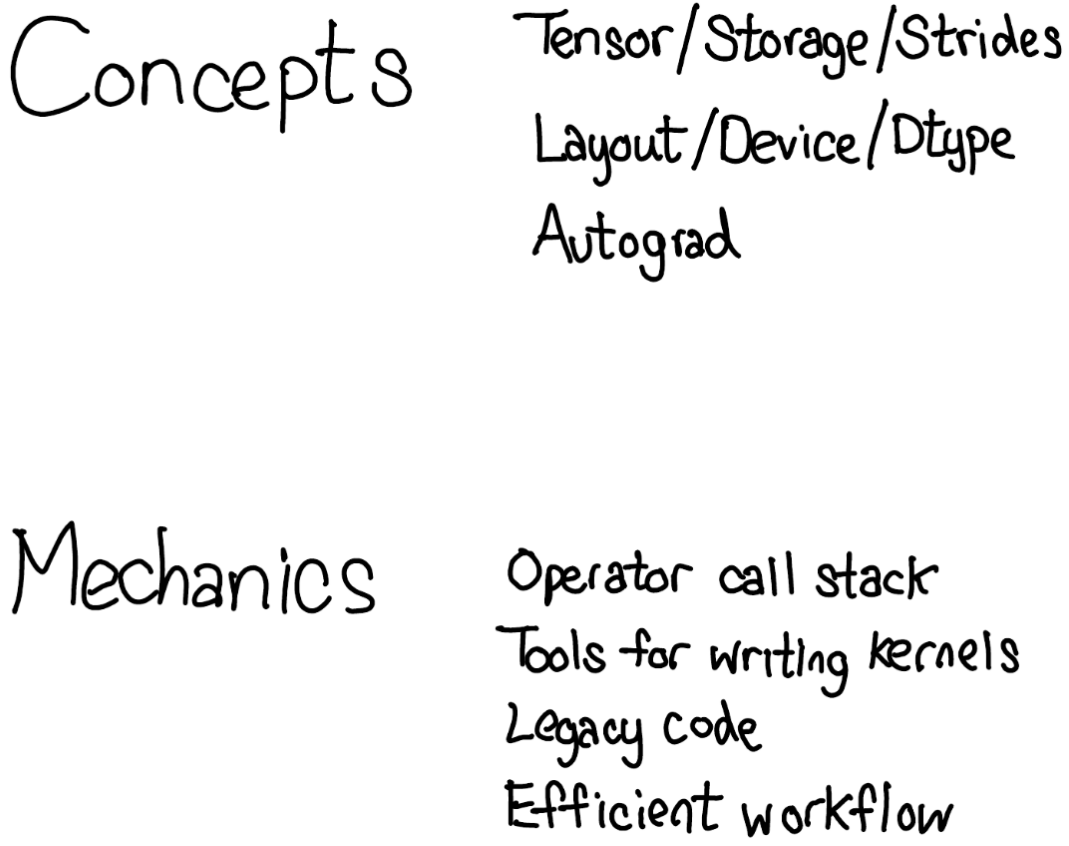


Image taken from blog-post with slides:
http://blog.ezyang.com/2019/05/pytorch-internals/

TensorFlow

TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API

TensorFlow

Numerical computation library
Express in Python
Underlying implementation: C++, CUDA

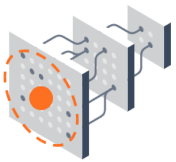
TensorFlow

Created by: Google Brain
Initial beta release: November 2015
Latest stable release: TF 1.13 April 2019 (TF 2.0 Alpha, May 2019)
License: Open source - Apache 2.0

Spiritual successor to Theano (2010) (discontinued 11/17)

Website: https://www.tensorflow.org/
Git: https://github.com/tensorflow/tensorflow

TensorFlow - Features



Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research

A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

from: https://www.tensorflow.org/

TensorFlow - Features

A summary of core features:

- **Data Flow Graphs**: describe mathematical computation with a directed graph of nodes & edges
- **Deep Flexibility:** if you can express your computation as a data flow graph, you can use TensorFlow
- **True Portability:** TensorFlow runs on CPUs or GPUs, and on desktop, server, or mobile computing platforms
- **Auto-Differentiation**
- **Maximize Performance:** Support for threads, queues & asynchronous computation. Freely assign compute elements to different devices
- **Estimators:** Has pre-defined set of commonly used estimators.

TensorFlow - Graph optimization

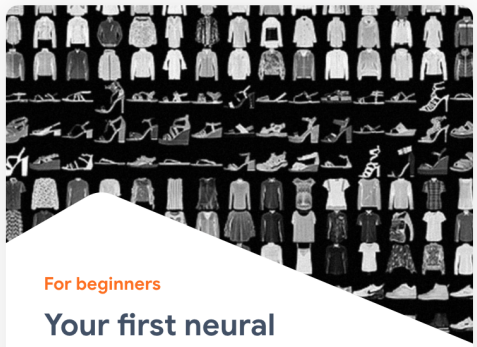
```
import tensorflow as tf
sess = tf.Session()
matrix1 = tf.constant([[3.],[3.]])
matrix2 = tf.constant([[3.],[3.]])
product = tf.matmul(matrix1,matrix2)
result = sess.run(product)
print(result)
sess.close()
```

Neural Network

```
import tensorflow as tf
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
def multilayer_perceptron(_X, _weights, _biases):
    layer_1 = tf.nn.relu(tf.add(tf.matmul(_X, _weights['h1']), _biases['b1']))
    layer_2 = tf.nn.relu(tf.add(tf.matmul(layer_1, _weights['h2']),_biases['b2']))
    return tf.matmul(layer_2, weights['out']) + biases['out']
...
# Initialize variables
# Launch the graph
```

TensorFlow - API

- Has additional layers API
- Previous auxiliary Keras library is now part of TensorFlow core. Keras defines a high-level API for models, optimizers etc.
- Now also has "eager execution" (imperative) mode
- Many tutorials available and large community



For beginners

Your first neural network

Train a neural network to classify images of clothing, like sneakers and shirts, in this fast-paced overview of a complete TensorFlow program.

For experts

Generative adversarial networks

Train a generative adversarial network to generate images of handwritten digits, using the Keras Subclassing API.

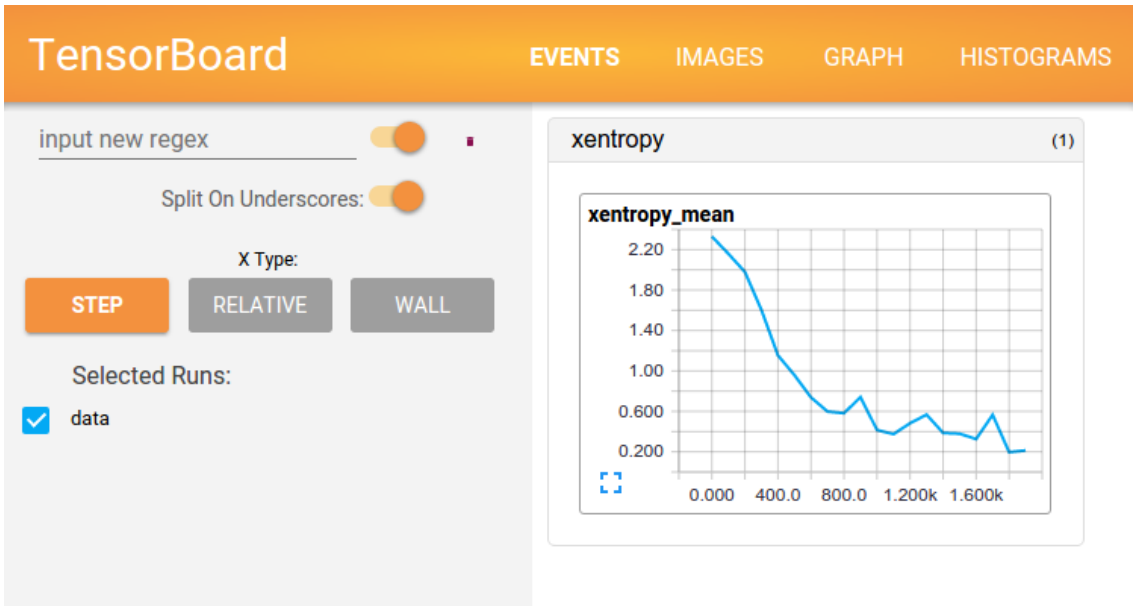
For experts

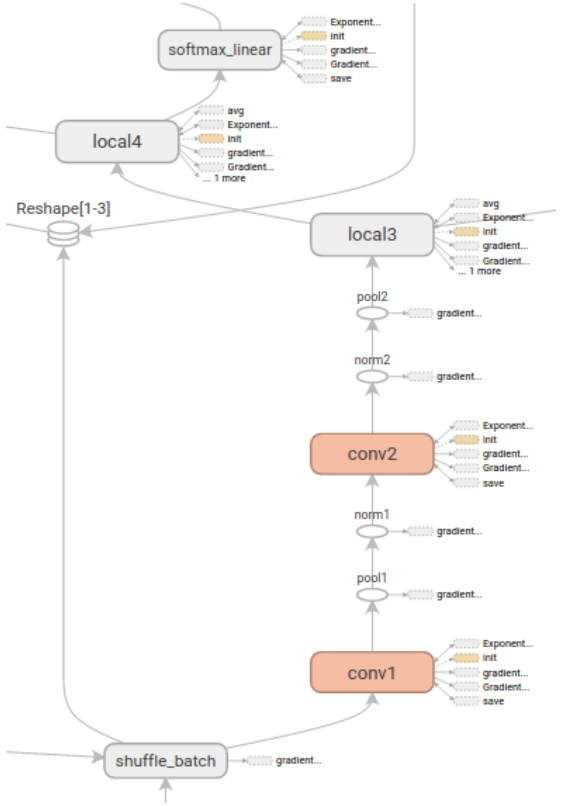
Neural machine translation with attention

Train a sequence-to-sequence model for Spanish to English translation using the Keras Subclassing API.

from: https://www.tensorflow.org/

TensorFlow





- Interactive playground: http://playground.tensorflow.org/
- Has been adapted for other frameworks

Honorable mentions of other ML/DL frameworks

Other frameworks (in no particular order):

- (Theano, Torch)
- MXNet
- Chainer
- Neon
- Caffe & Caffe2
- Deeplearning4j
- CNTK
- Model sharing ONNX

(GP)GPU computation in Machine Learning

- In principle multiple GPU vendors & software.
- In practice in ML almost exclusive application of Nvidia GPUs with CUDA.
- Very useful as large amount of operations in e.g. NNs are elementwise. Elementwise operations imply that the individual elements can be computed fully in parallel.
- Is particularly useful because one update in algorithms like SGD is typically based on a population of inputs. For these inputs (e.g. different images) the application of the complete pipeline can be calculated independently in parallel.
- Does not help with temporally correlated data or the necessary sequentiality of updates (relying on information of previous steps).
- Application typically limited by specific hardware constraints like memory limits.

GPU acceleration

	TITAN RTX	RTX 2080 Ti FE	TITAN V	TITAN Xp
Architecture	Turing - TU102	Turing - TU102	Volta - GV100	Pascal
Process	12nm FNN	12nm FNN	12nm	16nm
SMs	72	68	80	30
CUDA Cores	4608	4352	5120	3840
Tensor Cores	576	544	640	
Tensor FLOPS	130	114	110	
RT Cores	72	68		
Texture Units		272	320	
ROPs		88		96
Giga Rays	11 GR/s	10 GR/s		
RTX OPS		78 Trillion		
Compute	16.3 TFLOPS	14 TFLOPS	13.8 TFLOPS	12.1 TFLOPS
Base Clock	1350 MHz	1350 MHz	1200 MHz	1405 MHz
Boost Clock	1770 MHz	1635 MHz	1455 MHz	1582 MHz
Memory Size	24GB	11GB	12GB	12GB
Memory Type	GDDR6	GDDR6	HBM2	GDDR5X
Memory Clock	7000 MHz	7000 MHz	850 MHz	2582 MHz
Memory Interface	384-bit	352-bit	3072-bit	384-bit
Memory Bandwidth	672 GB/s	616 GB/s	652.8 GB/s	547.6 GB/s
Max. GPU Temp	89	89	91	94
Power Connectors	8+8 pin	8+8 pin	6+8 pin	6+8 pin
Graphics Card Power (TDP)	280W	260W	250W	250W

nvidia.com;
https://thepcenthusiast.com/nvidia-titan-rtx-vs-rtx-2080-ti-founders-edition-vs-titan-v-vs-titan-xp-specifications-comparison/

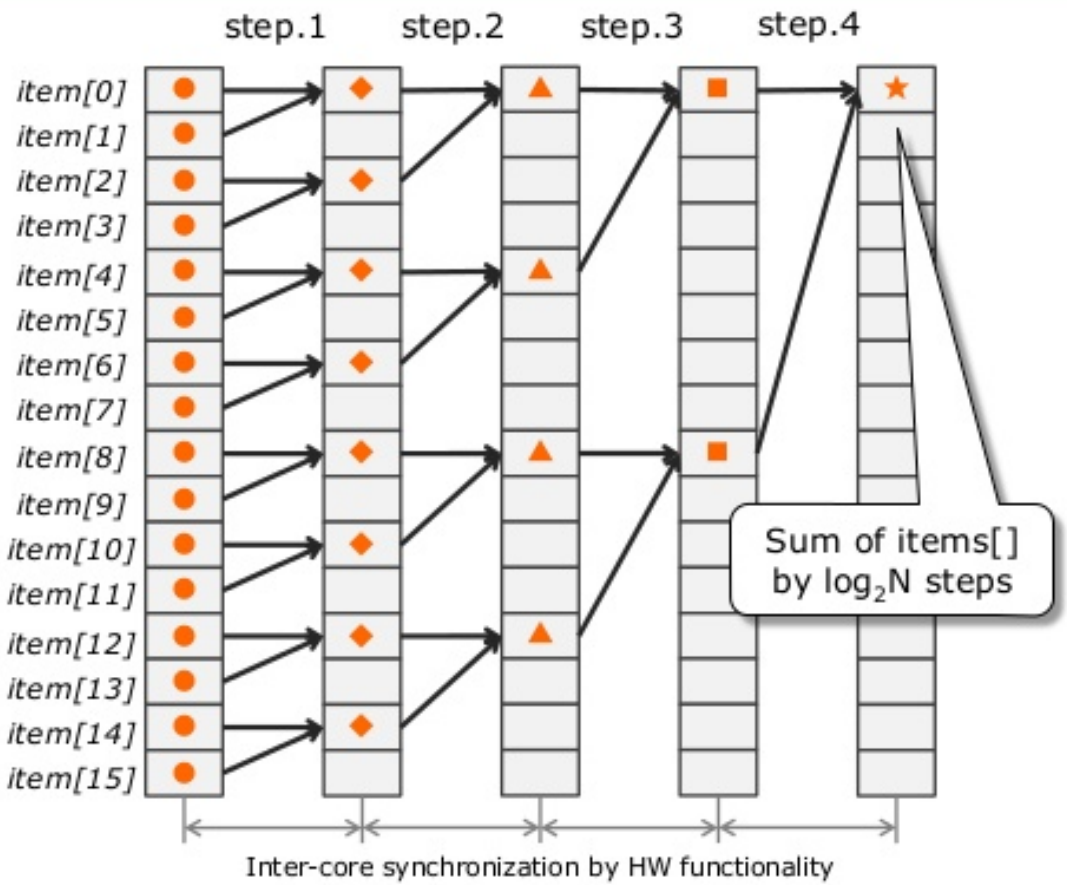
A non-trivial example

How GPU cores works

Calculation of

$$\sum_{i=0\ldots N-1} item[i]$$

with GPU cores



https://www.slideshare.net/kaigai/gpgpu-accelerates-postgresql-unlock-the-power-of-multithousand-cores

GPU acceleration in PyTorch

- Nvidia GPU with corresponding installed CUDA version

```
epochs = 5
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Model().to(device)

for e in range(epochs):
        for i, (inp, target) in enumerate(train_loader):
                inp = inp.to(device)
                target = target.to(device)

                output = model(inp)
                loss = criterion(output, target)

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
```