A short introduction to PyTorch

Stefan Duffner
LIRIS, INSA de Lyon

Introduction

- PyTorch: open source deep learning library for Python based on Torch
- Torch: machine learning library providing a flexible n-dimensional array structure (tensor), like numpy, with strong GPU acceleration
- Fast automatic differentiation system (autograd) used for gradient Backpropagation

Main Resources

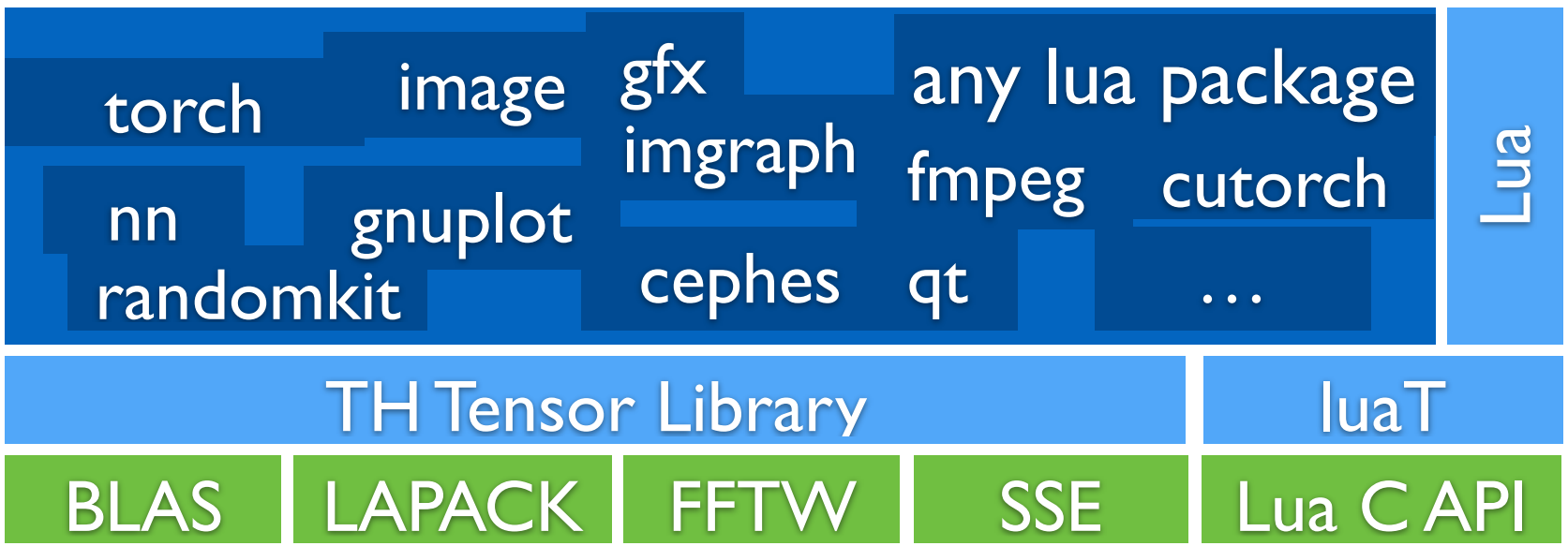
- Official web site: https://pytorch.org
- Documentation: https://pytorch.org/docs/stable/index.html
- Tutorials:
  - Deep Learning with PyTorch (60 min.)
  - Learning PyTorch with Examples

Torch7

- Main developer Ronan Collobert, since 2000
- Based on Lua / LuaJIT and C
- Why not python? Code is
  1. fast
  2. portable
  3. self-contained
- Large-scale machine learning
- Used by Google, Facebook, Twitter

Packages

- Many packages available
- Used by PyTorch (e.g. torch, nn, autograd)



Torch resources

- Main page: www.torch.ch
- github: https://github.com/torch
- demos/tutorials:
  https://github.com/clementfarabet/torch7-demos
- cheat sheet:
  https://github.com/torch/torch7/wiki/Cheatsheet

Back to PyTorch: Local installation

Get http link from www.pytorch.org! (select Linux/pip/no CUDA)

```
pip3 install --user --upgrade pip
pip3 install --user http://...
pip3 install --user torchvision
```

PyTorch: Tensors

```
from __future__ import print_function
import torch

a  = torch.empty(5, 3)
b = torch.rand(5, 3)
print(b)
print(b.size())
c = torch.tensor([5.5, 3])
d = torch.rand(5, 3, dtype=torch.double)
z = a + b
z = torch.add(a,b)
a.add_(b)
print(a[0,:])
b = a.numpy()
c = torch.from_numpy(b)
```

A first CNN (1/2)

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


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolu
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

A first CNN (2/2)

```
def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimensi
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Stimulate

```
net = Net()
print(net)

input = torch.randn(1, 1, 32, 32)  # nsamples x nchannels x h x w
output = net(input)
print(output)
```

Back-propagate

```
net.zero_grad()

target = torch.tensor([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]],
                      dtype=torch.float)
criterion = nn.MSELoss()
loss = criterion(output, target)
print(loss)
loss.backward()
```

Back-progagate and weight update made simple

```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

First exercise: image classification

- We will train a neural network to classify small colour images (32x32) into 10 different classes:

- CIFAR10
- airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
- *torchvision* module
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html