max planck institut
informatik

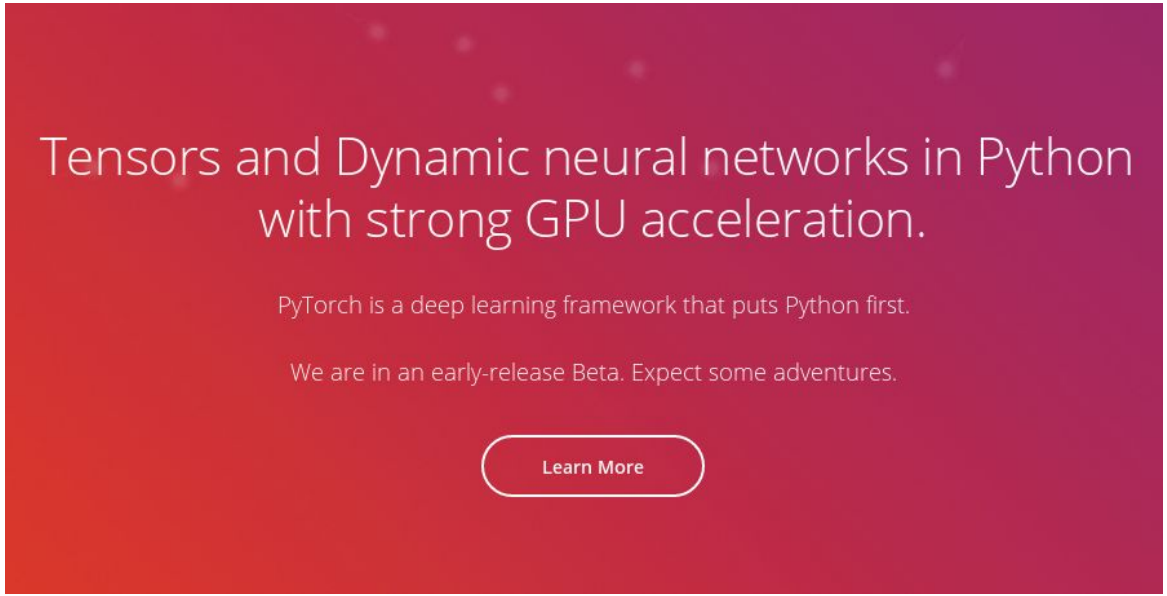
High Level Computer Vision

PyTorch - Quick Introduction

Rakshith Shetty - 14/07/2017

Some slides borrowed from:
http://dl.ee.cuhk.edu.hk/slides/tutorial-pytorch.pdf

What is it?



facebook · NVIDIA · salesforce · ParisTech · Carnegie Mellon University · Université Pierre & Marie Curie · Digital Reasoning · Stanford University · University of Oxford · NYU · Inria · ENS

What is it?

- A library that allows tensor based computation (like matlab/ numpy)
  - Easily run on GPU or CPU.
  - **Do automatic differentiation! Very useful for backpropagation**
  - One of the fastest (maybe caffe is a bit faster)
  - Several library functions which allows you to quickly

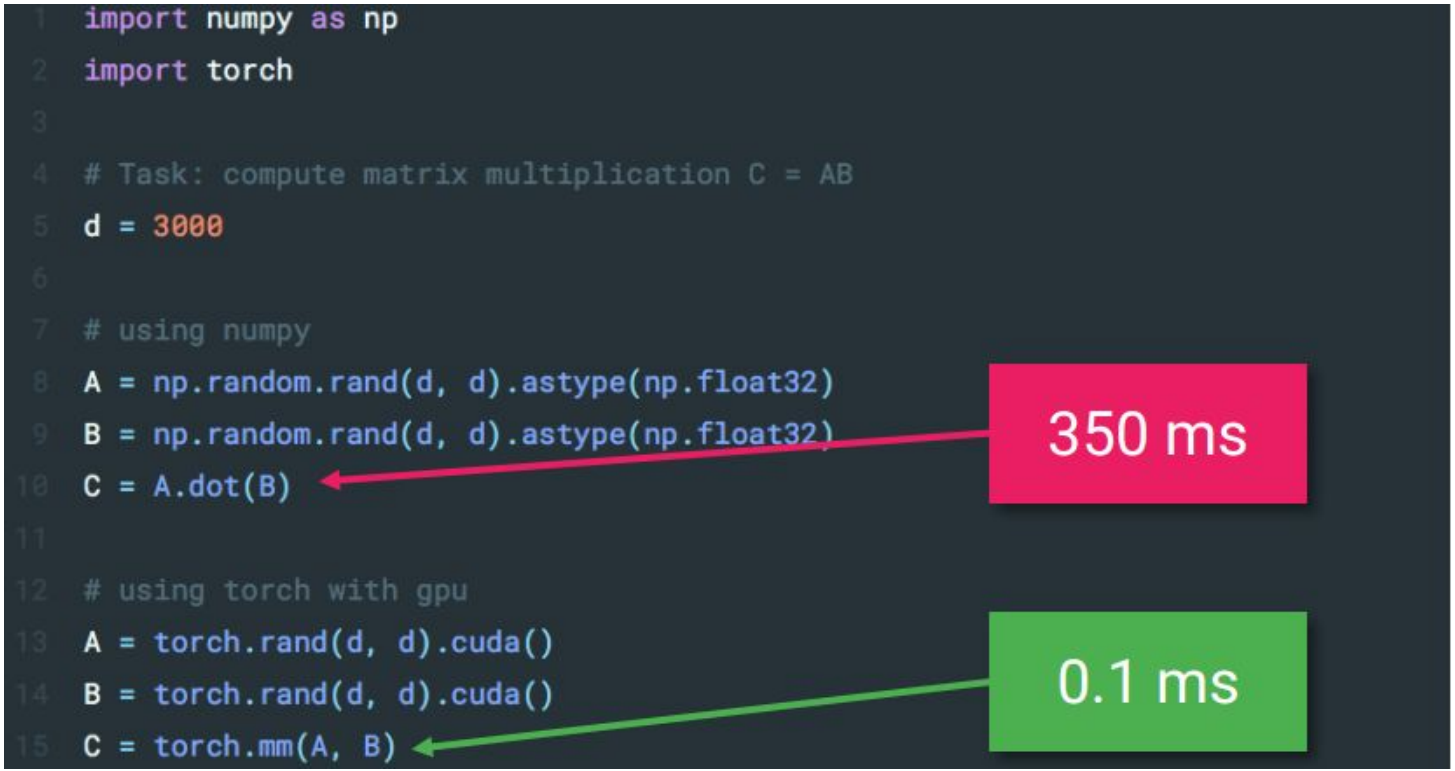
- What's different to other platforms?
  - **Dynamic computational graphs**
  - Very useful when dealing with recurrent networks or other wacky architectures

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

Basics


import numpy as np
import torch
# Task: compute matrix multiplication C = AB
d = 3000
# using numpy
A = np.random.rand(d, d).astype(np.float32)
B = np.random.rand(d, d).astype(np.float32)
C = A.dot(B)
# using torch with gpu
A = torch.rand(d, d).cuda()
B = torch.rand(d, d).cuda()
C = torch.mm(A, B)
350 ms
0.1 ms

Auto-differentiate

```
import torch
from torch.autograd import Variable

# Task: compute d(||x||^2)/dx
x = Variable(torch.range(1, 5), requires_grad=True)
print(x.data)  # x.data = [1, 2, 3, 4, 5]

f = x.dot(x)
print(f.data)  # f.data = 55

f.backward()
print(x.grad)  # x.grad = [2, 4, 6, 8, 10]
```

Auto-differentiate

```
import torch
from torch.autograd import Variable

# Task: compute d(||x||^2)/dx
x = Variable(torch.range(1, 5), requires_grad=True)
print(x.data)  # x.data = [1, 2, 3, 4, 5]

f = x.dot(x)
print(f.data)  # f.data = 55

f.backward()
print(x.grad)  # x.grad = [2, 4, 6, 8, 10]
```

Components

Package	Description
torch	a Tensor library like NumPy, with strong GPU support
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.
torch.multiprocessing	python multiprocessing, but with magical memory sharing of torch Tensors across processes. Useful for data loading and hogwild training.
torch.utils	DataLoader, Trainer and other utility functions for convenience

Sample Code

Create Layers

Initialize weights

Do forward computations

```
import torch.utils.data
import torch.nn as nn
from torch.autograd import Variable
from torch import tensor
import numpy as np

class MLP_classifier(nn.Module):
    def __init__(self, params):
        super(MLP_classifier, self).__init__()
        ##1 is to allow padding index
        self.output_size = params.get('num_output_layers',205)
        self.hid_dims = params.get('hidden_widths',[])
        self.inp_size = params.get('pca',-1)


        prev_size = self.inp_size
        self.hid_dims.append(self.output_size)
        self.lin_layers = nn.ModuleList()
        self.non_linearities = nn.ModuleList()
        self.dropouts = nn.ModuleList()
        for i in xrange(len(self.hid_dims)):
            self.lin_layers.append(nn.Linear(prev_size, self.hid_dims[i]))
            self.non_linearities.append(nn.ReLU())
            self.dropouts.append(nn.Dropout(p=params.get('drop_prob',0.25)))
            prev_size = self.hid_dims[i]

        self.softmax = nn.LogSoftmax()
        self.init_weights()
        # we should move it out so that whether to do cuda or not should be upto the user.
        self.cuda()

    def init_weights(self):
        # Weight initializations for various parts.
        a = 0.01
        # LSTM forget gate could be initialized to high value (1.)
        for i in xrange(len(self.hid_dims)):
            self.lin_layers[i].weight.data.uniform_(-a, a)
            self.lin_layers[i].bias.data.fill_(0)

    def forward(self, x, compute_softmax = False):
        x = Variable(x).cuda()
        prev_out = x

        for i in xrange(len(self.hid_dims)-1):
            prev_out = self.dropouts[i](prev_out)
            prev_out = self.non_linearities[i](self.lin_layers[i](prev_out))
        prev_out = self.dropouts[-1](prev_out)
        prev_out = self.lin_layers[-1](prev_out)

        if compute_softmax:
            prob_out = self.softmax(prev_out)
        else:
            prob_out = prev_out

        return prev_out
```

Useful resources

- Official documentation
  - http://pytorch.org/docs/
- Tutorials
  - http://pytorch.org/tutorials/
  - https://github.com/pytorch/tutorials
  - http://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (Useful)
- Example projects
  - https://github.com/pytorch/examples