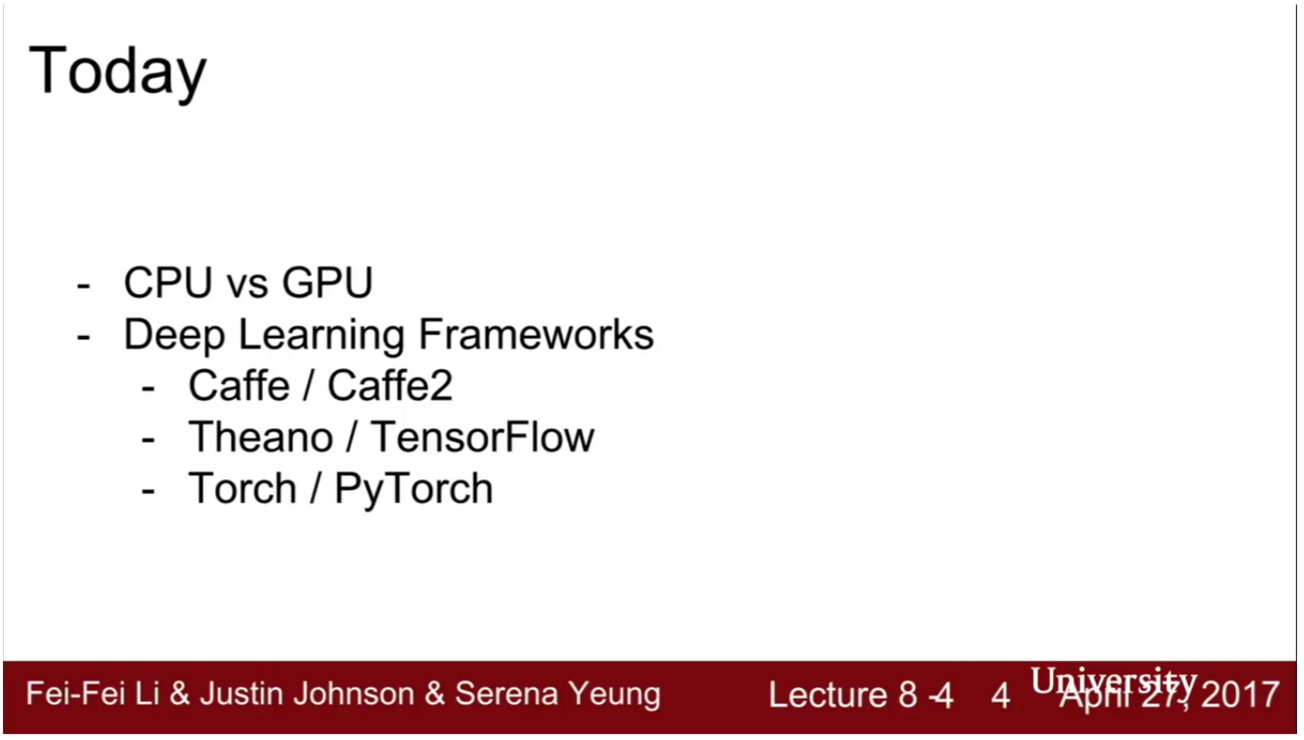
lec8

Creation Date: 10/01/2020 15:34

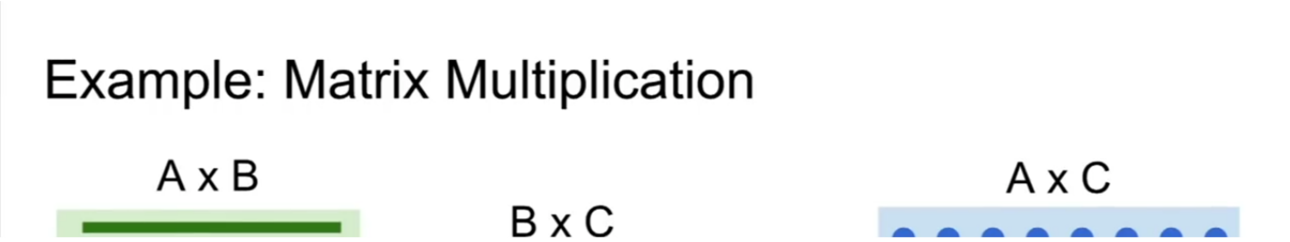
Last modified date: 10/01/2020 18:28

Lec 8 : Deep Learning Softwares

- CPU vs GPU
    - 

    - 

    - GPU cores cant run independently whereas CPU cores can.
    - Because of the large number of cores in GPU, they are prefered for parallel tasks.
    - CPU have cache, but they are relatively small in size.

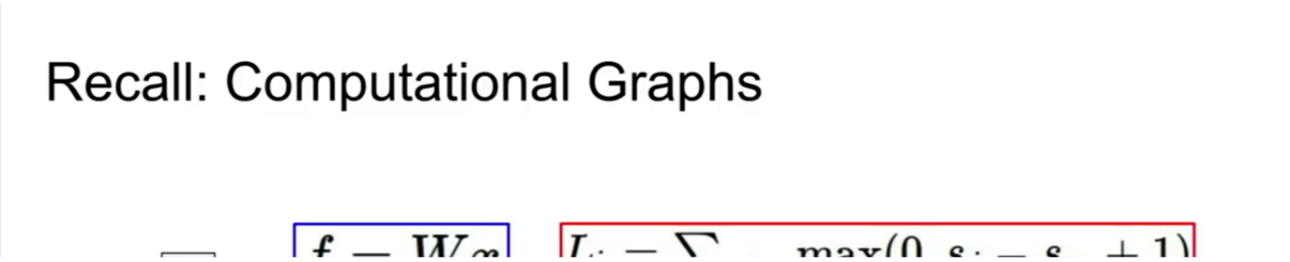


- Each dot product in the product matrix is independent of other entries.
- Since CPUs run instructions sequentially, matrix multiplication will be slower compared to the calculation done by GPUs.
- **Even Convolution operation can be parallelized on GPUs**

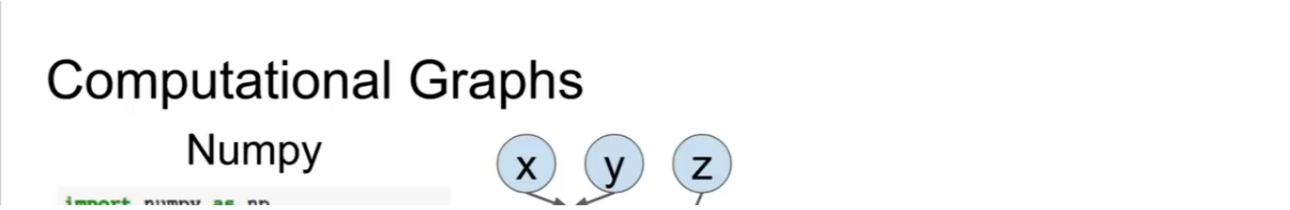
-
- **Note: USE cuDNN module, because it's almost 3x faster than 'unoptimized' CUDA.**

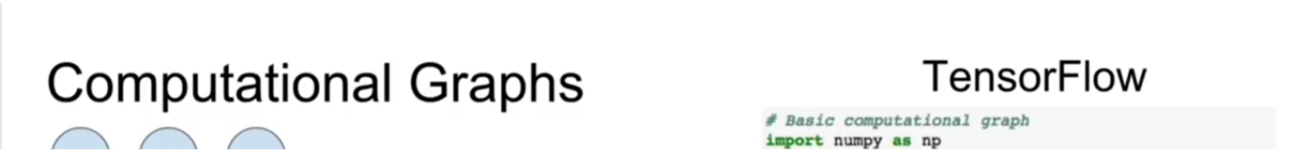
-

- Deep Learning Frameworks

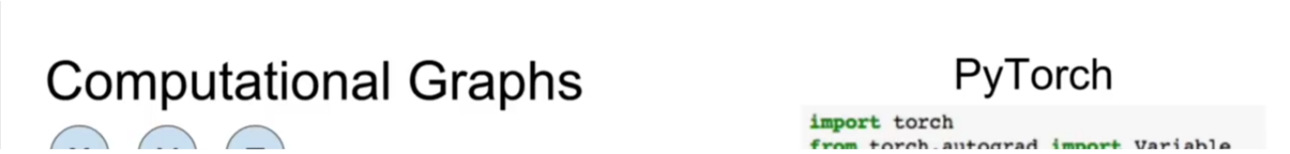
- 

-
- The point of deep learning frameworks:
  - Easily build big computational graphs
  - Easily compute gradients in computational graphs
  - Run it all efficiently on GPU (wrap around cuDNN, cuBLAS, etc)

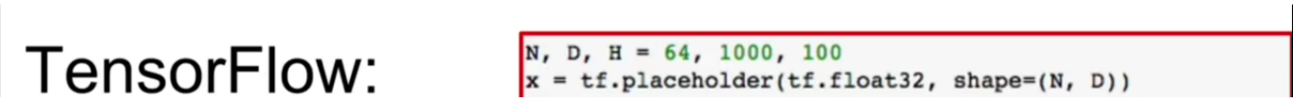


- 

- The tf.gradients function call will compute the gradients at each node.
- We can change between CPU and GPU by tf.device call.



-
- the Variable function creates a node in PyTorch
- the backward() function compute the gradients.
- We can use .cuda() or .cpu() to switch between gpu and cpu.



-
- tf.placeholders are just locations, no memory is allocated.
- until we create a session and run(), we are just creating the computational graph, there is no data in the system.
- We create the data and assign it to 'values'.
- in session.run() we pass the list of parameters whose value we want to compute.


TensorFlow: Distributed Version

-

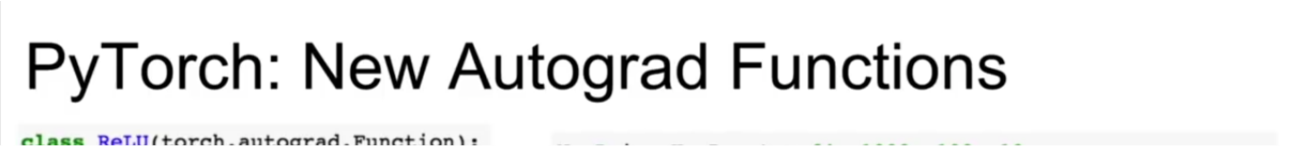

Side Note: Theano
import theano
import theano.tensor as T

-

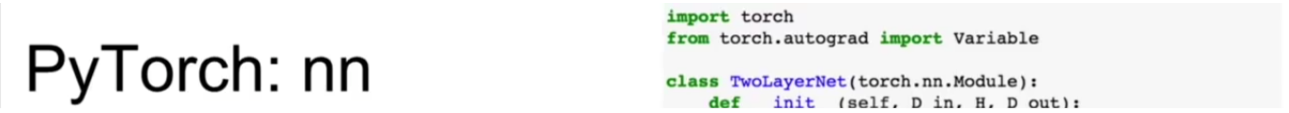
-
- the difference between numpy array and a tensor is that the tensor runs on a GPU.

-

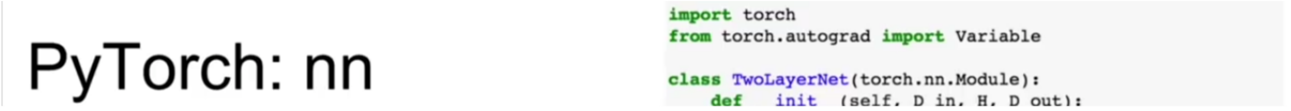
- **In PyTorch we create the graph everytime we do the forward pass.**


PyTorch: New Autograd Functions
class ReLU(torch.autograd.Function):

○


PyTorch: nn
import torch
from torch.autograd import Variable
class TwoLayerNet(torch.nn.Module):
def __init__(self, D_in, H, D_out):

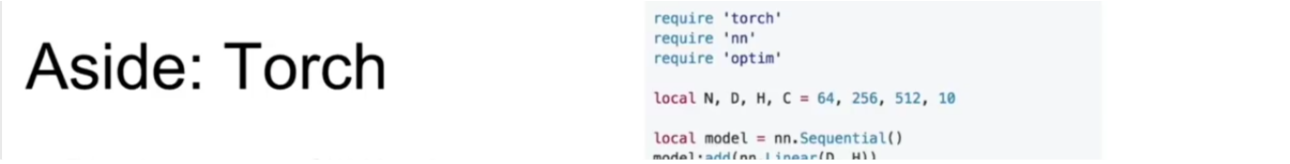
○


PyTorch: nn
import torch
from torch.autograd import Variable
class TwoLayerNet(torch.nn.Module):
def __init__(self, D_in, H, D_out):

◦


PyTorch: DataLoaders
import torch
from torch.autograd import Variable

◦


Aside: Torch
require 'torch'
require 'nn'
require 'optim'
local N, D, H, C = 64, 256, 512, 10
local model = nn.Sequential()


Static vs Dynamic Graphs
TensorFlow: Build graph once, then
PyTorch: Each forward pass defines

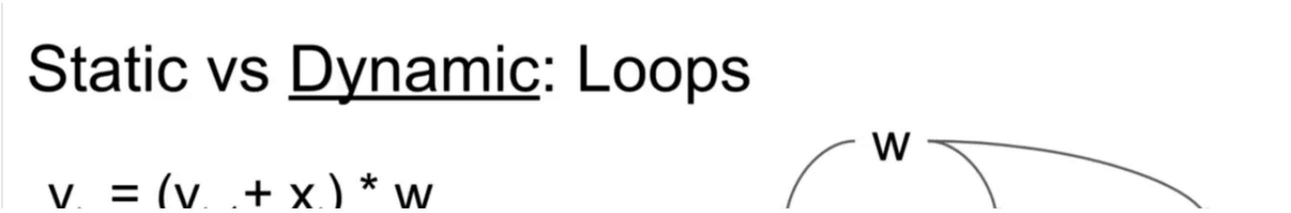
Static vs Dynamic: Optimization

With static graphs The graph you wrote Equivalent graph with

Static vs Dynamic: Serialization

Static **Dynamic**


Static vs Dynamic: Conditional
TensorFlow: Special TF


Static vs Dynamic: Loops
w

Dynamic Graphs in TensorFlow

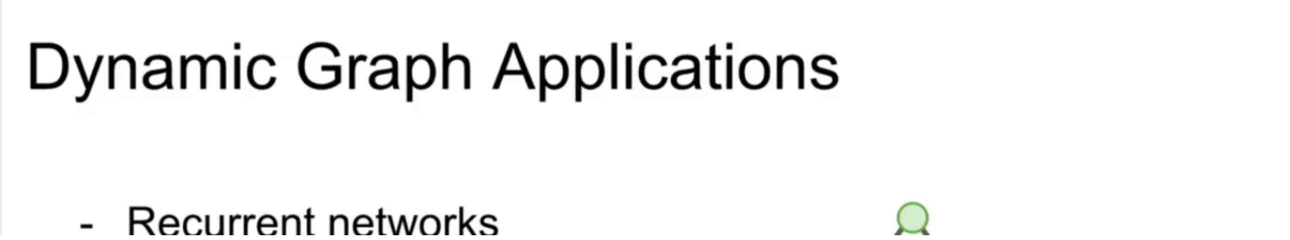
TensorFlow Fold make dynamic

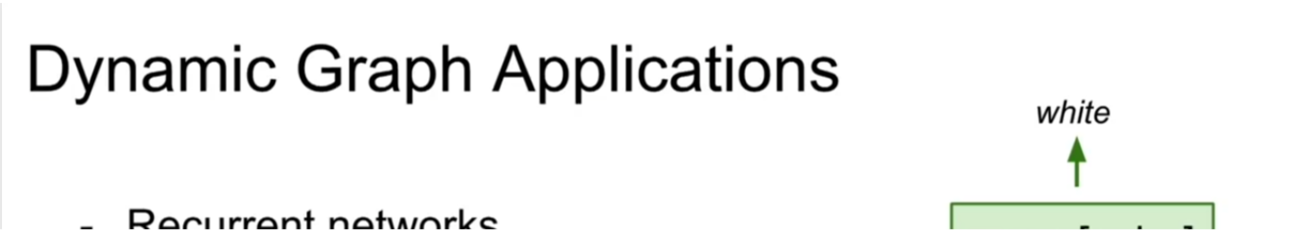
○

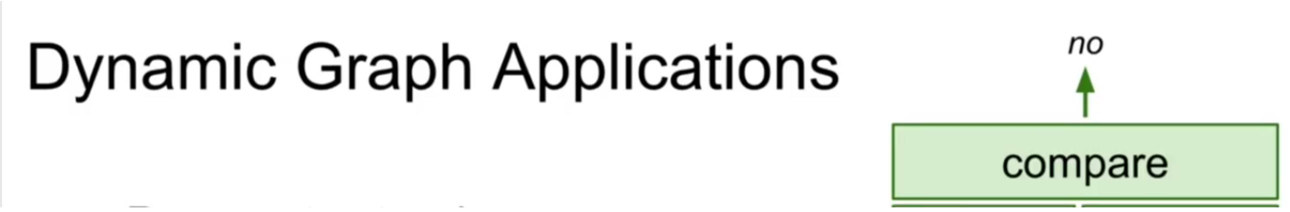
Dynamic Graph Applications

- Recurrent networks

○


Dynamic Graph Applications
- Recurrent networks


Dynamic Graph Applications
white
- Recurrent networks


Dynamic Graph Applications
no
compare


Caffe Overview
● Core written in C++
● Has Python and MATLAB bindings

Caffe step 1: Convert Data

- DataLayer reading from LMDB is the easiest

○

Caffe step 2: Define Network (prototxt)

```
name: "LogisticRegressionNet"
```

○

Caffe step 2: Define Network (prototxt)

```
layer {
        bottom: "res5c"
```

Caffe step 3: Define Solver (prototxt)

Caffe Pros / Cons

- (+) Good for feedforward networks

Caffe2 Overview

- Very new - released a week ago =)


Google:
TensorFlow
Facebook:
PyTorch +Caffe2


My Advice:
Subtitle track: Disable