IVU LECTURES

PYTORCH BASIC KNOWLEDGE

FACTS ON TENSORS, MODULES AND FUNCTIONALS FOR YOUR DAILY DEEP LEARNING ROUTINE

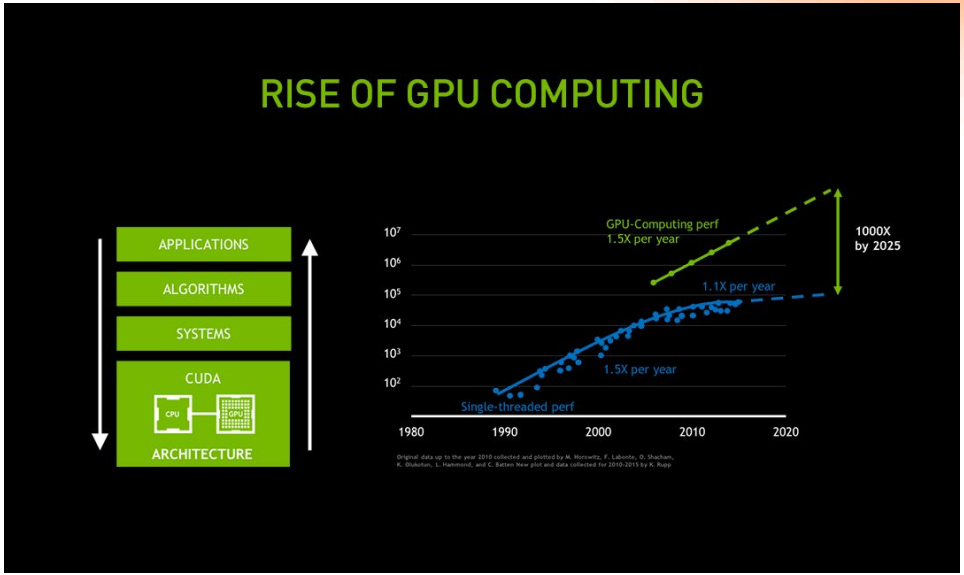
PYTORCH

PyTorch is a Python library mainly used to perform deep learning tasks such as building, training and evaluating neural network models.

GPU-Accelerated Computing

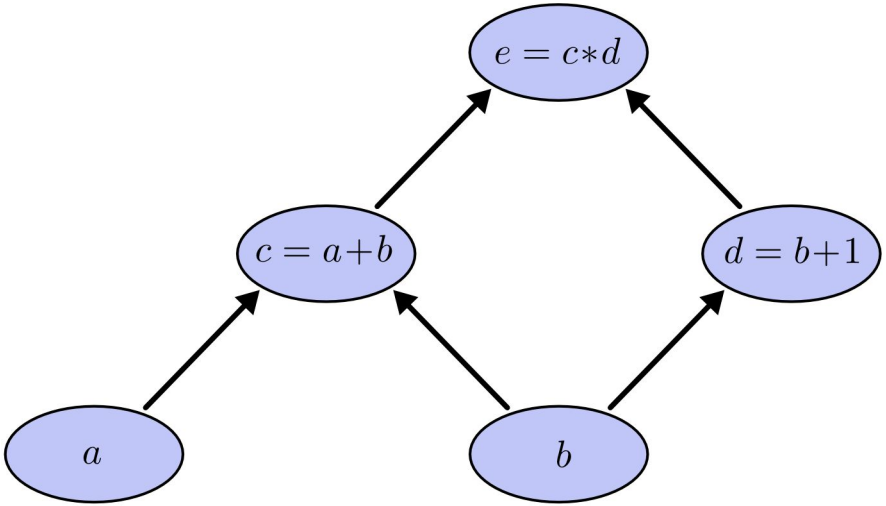
PyTorch is deeply integrated with CUDA, to run the training of models directly in the GPU

- Tensors and models can be sent to the GPU
- Gradient checkpointing can be used if GPU memory is slow
- GPU acceleration dramatically improves the time performances



PYTORCH

PyTorch is a Python library mainly used to perform deep learning tasks such as building, training and evaluating neural network models



Dynamic Derivative Graph

PyTorch allows user to change dynamically the model structure at runtime:

- Network defined in a modular way. Modules can be deleted and added
- Useful for RNNs/LSTMs and model pruning
- Require the user to manually zero out gradients

PYTORCH

Its father

- Lua-based framework with the same purposes
- Inspirational source of Pytorch and the two frameworks share some authors
- **Not actively supported anymore**

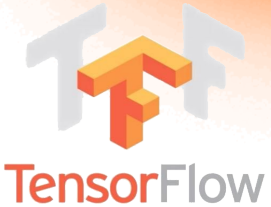
Its mother

- Python-based framework with the same purposes
- Has dynamic graphs too
- Uses autograd too
- **Lot of backend in Python**
- **Worse efficiency**

Its cousin

- Python-based MATLAB-like library for scientific computing
- Popular tensor indexing and slicing mechanics
- Lots of scientific computing algorithms (check also SciPy, scikit-learn, matplotlib)
- **Poor GPU support for Deep Learning tasks**

Its rival



- Python-based too
- Older than PyTorch
- Largely used in production environments
- **Harder to debug**
- **Default static differentiation graph**
- Will switch to a PyTorch-inspired framework within months

Basic structures

Tensor: the most important one. It's used to perform array-like computations i.e. vectorial sums, vectorial/element-wise products, etc.

Characteristics

- Very similar to NumPy's *ndarray* structure (efficient NumPy conversion)
- Advanced indexing/slicing (since v0.3)
- Can store associated derivatives values:
  - v0.4: *requires_grad/requires_grad_()*
  - v0.3: *Variable* wrapping
- Can be sent to GPU for faster computations:
  - v0.4: *to(device='gpu:0')*
  - v0.3 *cuda()*

Basic structures

Module: base class used to create NNs building blocks. It is used to represent convolutional operators, non-activation function, losses, grouping of Modules, etc.

Characteristics

- Defines a *forward* abstract method, for the forward pass
- *forward* is then called by *__call__* method
- Has methods to easily debug gradients (*register_*_hook*)
- It usually stores a *Functional* performing the operation (not covered in this tutorial)

Parameter: subclass of Tensor used for model weights that need to be updated.

Characteristics

- Each *Parameter* field of a Module object will receive gradient updates during the backward pass

Basic structures

_Loss: Module subclass which represents losses. Includes MSE, cross-entropy, etc.

Characteristics

- Calling *backward* method will results in computing all the needed gradients.

Optimizer: Base class for optimizers. Includes SGD, Adam, etc.

Characteristics

- Accept a group of arguments (*lr*, *momentum*, *weight_decay*, etc.) and a group of weights to be updated (just call *Module.parameters* method on your model)
- Call *step* method to start the weights update

Basic structures

Dataset: base class representing a train or test dataset.

Characteristics

- Two methods must always be present:
  - *__len__* to retrieve the length of the dataset
  - *__getitem__* to get items from the dataset.
- Lots of standard datasets are already defined in *torchvision* library

DataLoader: base class which automatizes the batches generation from Dataset objects.

Characteristics

- Requires a *Dataset* object to iterate on
- Usually the standard *DataLoader* suffices for classification tasks, however more specialized ones are defined
- It is used in for cycles. *__iter__* method returns an iterator on the *Dataset* object

@googleslides

LET'S START