Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Timeline

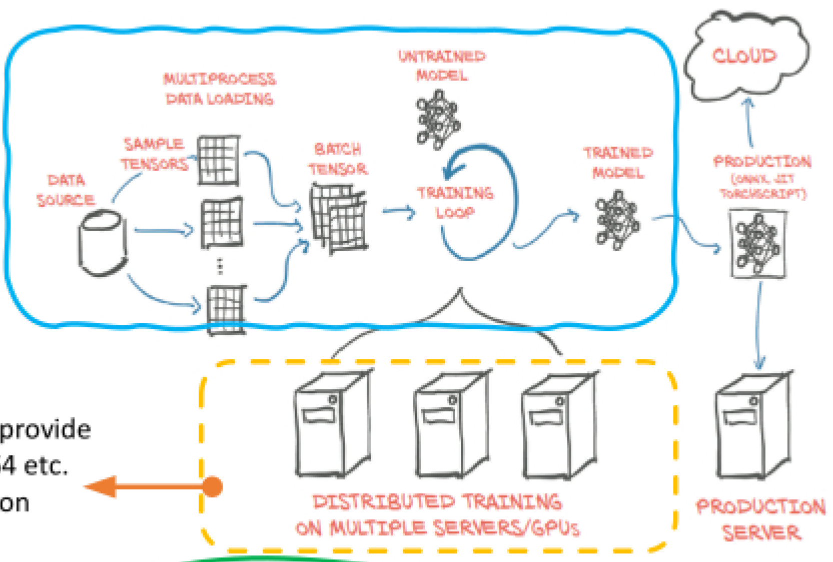
- Quiz 2: March 30th, Tuesday, 10:00am – 11:15am
- Proposal submission due on March 26th, Friday.
- University Calendar: No school on April 2nd, Friday.
- Review proposals: April 6th, Tuesday.

What is PyTorch?

- PyTorch is a Machine Learning library built on top of torch. It is backed by Facebook's AI research group. After being developed recently it has gained a lot of popularity because of its simplicity, dynamic graphs, and because it is pythonic in nature. It still doesn't lag behind in speed, it can even out-perform in many cases.
- Python-based scientific computing library, similar to NumPy.
- Differentiates from NumPy in 3 main aspects:
  - It allows to use the power of GPU computing
  - It comes with an automatic differentiation module
  - It is a fully-fledged deep learning research platform
- PyTorch can work well on multi-GPU and/or multi-node comparing with other DL frameworks.

https://pytorch.org/

Deep Learning pipeline [with PyTorch]

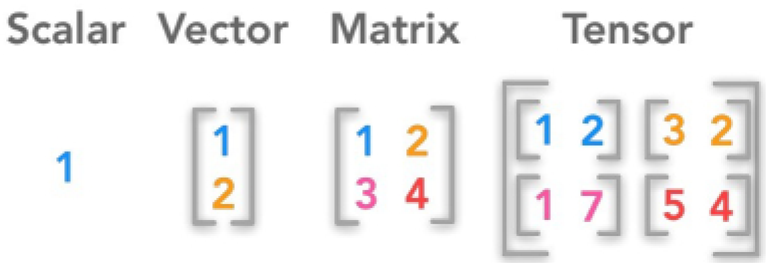

MULTIPROCESS DATA LOADING
UNTRAINED MODEL
CLOUD
SAMPLE TENSORS
DATA SOURCE
BATCH TENSOR
TRAINING LOOP
TRAINED MODEL
PRODUCTION (ONNX, JIT TORCHSCRIPT)
DISTRIBUTED TRAINING ON MULTIPLE SERVERS/GPUs
PRODUCTION SERVER

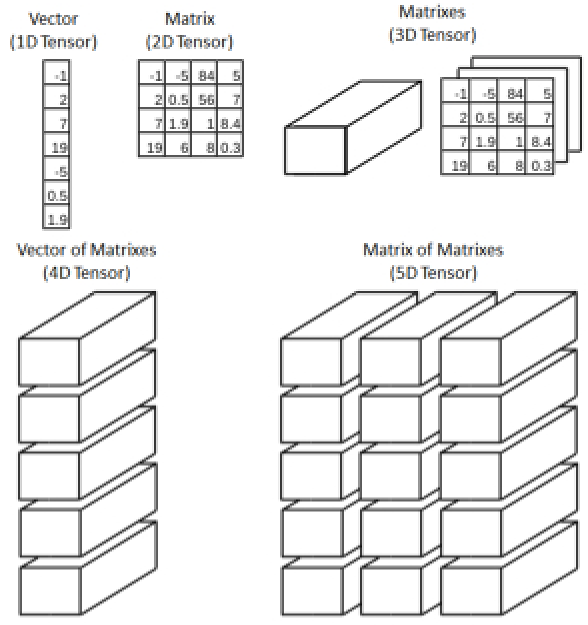
- GPU Tensor Cores: provide FP16, FP32 and FP64 etc.
- Parallel computing on multiple GPUs

It's all about Tensors!

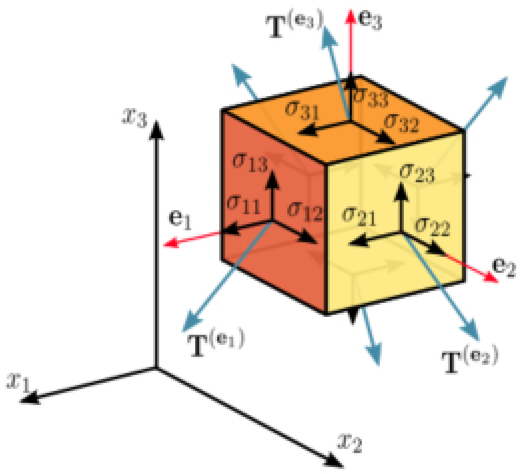
The illustration is taken from PyTorch.

Quick ML/DL Concepts: Tensors




Vector (1D Tensor)
Matrix (2D Tensor)
Matrixes (3D Tensor)
Vector of Matrixes (4D Tensor)
Matrix of Matrixes (5D Tensor)

Tensor in Deep Learning and Neural Networks



Tensors can be best viewed as containers that wrap and store data features in the context of machine learning and deep learning.

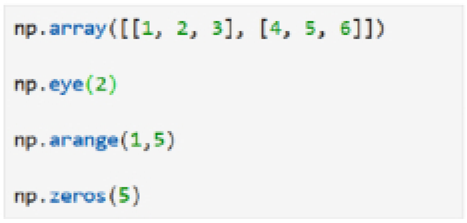
N-Dimension Tensors in DL

- To give you an idea of how often tensors are used in the world of Deep Learning, the most common types of tensors are:
  - 3D tensors: used in **time series**.
  - 4D-Tensors: used with **images**.
  - 5D tensioners: used with **videos**.

- Normally, one of the dimensions will be used to store the samples of each type of data.

Tensors in PyTorch

- Tensors are multi dimensional arrays
- Very similar to NumPy for Tensor creation, indexing, masking
- PyTorch vs NumPy



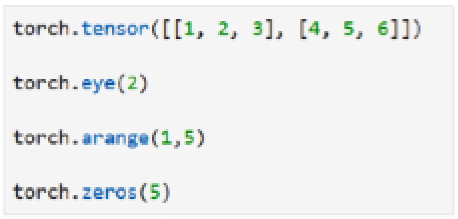
```
np.array([[1, 2, 3], [4, 5, 6]])

np.eye(2)

np.arange(1,5)

np.zeros(5)
```

$\longrightarrow$



```
torch.tensor([[1, 2, 3], [4, 5, 6]])

torch.eye(2)

torch.arange(1,5)

torch.zeros(5)
```

Automatic Differentiation

- Automatic differentiation is a key feature of PyTorch.
- PyTorch can differentiate the outcome of any computation with respect to its inputs.
- This allows you to express and optimize complex models without worrying about correctly differentiating the model.

Computing Gradients with Autograd

- Autograd: Automatic Differentiation package
- torch.autograd is PyTorch’s automatic differentiation engine that powers neural network training.
- Each Tensor has a requires_grad boolean attribute
- Autograd creates a graph to record all operations during the computation
- Call tensor.backward() to compute all gradients automatically
- Gradients are accumulated into the tensor.grad attribute

Background

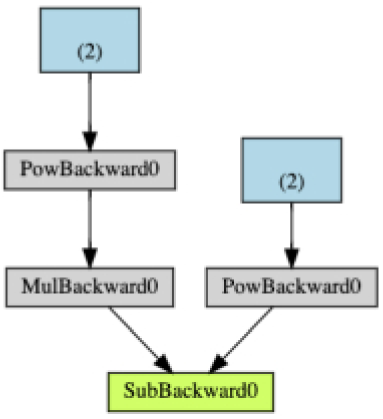
- Training a NN happens in two steps:
  - **Forward Propagation**: In forward prop, the NN makes its best guess about the correct output. It runs the input data through each of its functions to make this guess.
  - **Backward Propagation**: In backprop, the NN adjusts its parameters proportionate to the error in its guess. It does this by traversing backwards from the output, collecting the derivatives of the error with respect to the parameters of the functions (gradients), and optimizing the parameters using gradient descent.

Computational Graph

- Conceptually, autograd keeps a record of data (tensors) & all executed operations (along with the resulting new tensors) in a directed acyclic graph (DAG) consisting of *torch.autograd.Function* objects.
- In a forward pass, autograd does two things simultaneously:
  - run the requested operation to compute a resulting tensor, and
  - maintain the operation's gradient function in the DAG.
- The backward pass kicks off when .backward() is called on the DAG root. autograd then:
  - computes the gradients from each .grad_fn,
  - accumulates them in the respective tensor's .grad attribute, and
  - using the chain rule, propagates all the way to the leaf tensors.

DAG

$$Q = 3 * a^3 - b^2$$



DAGs are dynamic in PyTorch An important thing to note is that the graph is recreated from scratch; after each *.backward()* call, autograd starts populating a new graph. This is exactly what allows you to use control flow statements in your model; you can change the shape, size and operations at every iteration if needed.

- Stay safe!
- See you next class!

<u>Next Lecture will Continue:</u>

High Performance PyTorch DL on GPU