Caffe, Caffe2 + Pytorch

Curtis Isaacson
Schedule

- **Caffe**
  - Introduction
  - History
  - The Need for Caffe
  - Highlights
  - Implementation
  - Model Implementation Demo

- **Caffe2**
  - Features and Improvements
  - Model Zoo

- **PyTorch**
  - Strengths
  - Integration with Caffe2
  - PyTorch 1.0
Introduction

- Convolutional Architecture for Fast Feature Embedding (Caffe)
- Deep Learning Framework designed modularity and speed in mind
- Enabled developers to build and share networks easily
- Yangqing Jia et al., Caffe: Convolutional Architecture for Fast Feature Embedding (Caffe)
  https://arxiv.org/abs/1408.5093
History and Background

- Published at Berkeley Vision and Learning Center in 2014
- As we know, the 2010’s saw a rise in the use of Deep Learning
  - Allowed programmers to skip feature engineering, which was particularly useful for difficult-to-represent data such as
    - Images
    - Sound
    - Haptics (Touch)
History and Background

- “Particularly Motivated by large-scale visual recognition”
- Computer Vision has been driven forward with the use of **Convolutional Neural Networks (CNNs)**
- Similar to basic Neural Networks however CNNs are “trained via back-propagation through layers of convolutional filters and other operations”
History and Background

● **Problem?**
  ○ Neural Nets are powerful but have an equal downside of being very complex
  ○ Neural Nets are difficult to share with others

● **This leads to issues within the distribution and training of these networks**
  ○ How do other researchers replicate our results?
  ○ How do we share our networks?
  ○ How do we conduct our own research by building upon the results of others?
  ○ How do we easily integrate our NN model with a current commercial applications?

● **Thus, Caffe was created to solve these questions**
  ○ Offers out of the box deployment of current models while retaining computational efficiency!
History and Background

- Caffe is an open source framework
- Written in C++
  - Very efficient!
- Supported in Python/Numpy and MATLAB
- CUDA enabled
  - Compute Unified Device Architecture (CUDA)
  - Created by Nvidia for computationally heavy applications
  - Allows for GPU’s that are CUDA enabled to be used for General Purpose Processing

Source: https://developer.nvidia.com/about-cuda
Highlights

- **Modularity**
  - Designed to be as modular as possible
  - Easy Extension for new
    - Data formats
    - Network Layers
    - Loss functions

- **Separation of Representation and Implementation**
  - Only reserves as much memory as it needs for the network
  - Can switch between CPU and GPU implementation very easily

- **Test Coverage**
  - Caffe Modules have been extensively tested, which allows for peace of mind when conducting research

- **Pre Trained Reference Models**
  - Popular and groundbreaking models ready for distribution
How does Caffe Data Storage work?

- LevelDB Databases
  - Open source, on-disk, key-value data store
  - 150 MB/s throughput with Minimal CPU impact, which is highly efficient

- Blobs
  - 4-dimensional arrays
  - “Blobs provide a unified memory interface, holding batches of images (or other data), parameters, or parameter updates.”

Figure 1: An MNIST digit classification example of a Caffe network, where blue boxes represent layers and yellow octagons represent data blobs produced by or fed into the layers.

https://caffe.berkeleyvision.org/tutorial/net_layer_blob.html
How does Caffe Data Storage work?

- Blobs allow for highly complex synchronized CPU and GPU operations, without the hassle of trying to handle those operations yourself.

- For example:
  - CPU loads data from the disk to a blob in CPU code,
  - Calls a CUDA kernel to do GPU computation
    - Remember how CUDA enables General Processing GPU functionality?
  - Ferries the blob off to the next layer, ignoring low-level details while maintaining a high level of performance.

- Essentially allows for the abstraction of data read and write operations, allowing for heightened performance.
How do Caffe Layers work?

- Remember question on how do we share our models?
- Caffe does so cleanly by creating a file that defines network architecture
- This is defined in plaintext protocol buffer schema (prototxt)
Simple Demo of Caffe in MATLAB
Caffe 2

- “As new computation patterns have emerged, especially distributed computation, mobile, reduced precision computation, and more non-vision use cases, Caffe’s design has shown some limitations”
- Facebook, after taking Caffe under its wing, added more functionality, and created Caffe2. This improved multiple things
  - Better support for large-scale distributed training
  - Mobile deployment
  - New hardware support (on top of CPU and CUDA)
  - Tested more rigorously at FB

https://caffe2.ai/docs/caffe-migration.html
Caffe 2 Model Zoo

- Caffe2 has a lot of really smart devs working on and with it!
- The Model Zoo is a place where you can share and use pre-trained models
  - Allows you to easily build a demo application
  - Sadly you have to convert models from Caffe to Caffe2 specific formats
- Basically just a big model repository on GitHub
- [https://github.com/facebookarchive/models](https://github.com/facebookarchive/models)
- Similar to ONNX
  - Open Neural Network Exchange
PyTorch

- Similar to Caffe, Open Source Machine Learning Library
- Written in Python
- Based on the Torch Library
- Use cases are very modern
  - Tesla Autopilot
  - Uber Pyro
- Can be used on major cloud platforms
- Easy Debugging
- Very heavily used for Research due to these features

PyTorch Distributed Training

- **Distributed Data-Parallel Training**
  - Implements data parallelism at the module level which can run across multiple machines.
  - Takes care of gradient communications to keep model replicas synchronized
  - Overlaps it with the gradient computations to speed up training

- **Collective Communication APIs**
  - `All_reduce` and `All_gather`

- **Several Different Options for Data Parallel Training**
  - Single-device training: assuming data and model can fit in one GPU, and you're not worried about speed
  - Single Machine, multi-GPU: if there are multiple GPU's on the server, you can speed up training with small code tweaks
  - Single Machine, multi-GPU in DistributedDataParallel: more code, but allows for greater speed up
  - Multiple Machine DistributedDataParallel: Utilize TorchElastic along with DistributedDataParallel
Data Parallel (DP) vs Distributed Data Parallel (DDP)

- DP is single process, multi thread, and only works on a single machine.
- Since DP tends to be simpler, easier to use out-of-the-box, it tends to be slower than DDP.
- If a model is too large to fit on a single GPU, you must use Model Parallel to split across multiple GPUs.
  - Intuitively, that means that single GPU systems wouldn't work.
- Model Parallelism is currently supported by DDP at this time.
- If model doesn't fit into Data Parallelism paradigm, you can use Remote Procedure Call Distributed Training, which PyTorch has functionality for.
- Examples of when it doesn’t fit include:
  - Distributed pipeline parallelism
  - Parameter server paradigm
  - Combinations of DDP with other training paradigms
When doing distributed training, what are some ideas you should think about?
- Keeping all processes synchronized
- What happens if a failure happens during training?
- How do I best handle a failure when training?

Torch Elastic steps in to help handle those errors
- Common errors like Out Of Memory (OOM) can plague Distributed Data Parallelism
- DDP itself cannot recover from those errors!
  - Requires all processes to be closely synchronized, and if one goes out of sync, things can go bad
  - This is due to AllReduce Communications that happen between systems

TorchElastic allows training jobs to be executed in a fault tolerant and elastic manner
- Fault Tolerance: jobs running on infrastructure where Hardware gets replaced often
- Dynamic Capacity Management: Jobs running on leased capacity that can change (AWS)
Caffe

- Specifically Caffe2
- Created for Production, Speed
  - Written in C++
- Focused on scalable systems as well as cross platform support
- Model Deployment is friendlier for Mobile Platforms

- Created for Research
- Focused on Research Flexibility
- Predominantly uses Python
- Research tends to have lots of debugging, parameter and model changes
  - PyTorch handles this better

https://analyticsindiamag.com/pytorch-vs-caffe2-which-machine-learning-framework-should-you-use-for-your-project
PyTorch 1.0

- Seeing as there are quite a few differences between the two, what would be the best course of action?
- May 2018
  - Facebook, and now-Facebook AI Infrastructure Director Yangqing Jia, decide to merge both PyTorch and Caffe2 together!
- Now we are able to get the best of both worlds.
  - Better Research Flexibility for the Scientists
  - Better Scalability and Cross Platform Support for the Business
Questions?