Deep Learning Libraries

Ujjwal

STARS team, INRIA Sophia Antipolis

December 7, 2017

Overview

1. First things first
2. Selecting your library
   - Selection Criteria
3. Getting comfortable with a library
4. Vital random musings
5. Fine-Tuning

First things first

A common agreement : DL libraries are complex.

A barely usable DL library (*imagine training/testing a 2-layer CNN*) design requires -

- Exhaustive knowledge of CPU & GPU architectures (hardware details !!)
- Command over memory management.
- Exquisite control over algorithmic complexity (esp. matrix computations.)

Almost every major DL library today has its basecode authored by a team pushed by a technical organization (e.g- Google, Facebook, Microsoft, Sony, Baidu)

Some terminology for the presentation

- **Model**
  - A complete end-to-end system performing a well-defined vision task (e.g- FRCNN (pedestrian detection), FCNN (segmentation))
- **Network**
  - A neural network consisting of convolutional or recurrent layers or both which extracts features from an image. (e.g- VGG16, Alexnet, Inception)
- **Inference**
  - The sequence of computations done by a *network* in order to compute its output.
- **Training**
  - Updating of weights in a *network* by repeated inference and backpropagation.
- **Forum**
  - Any online website where programming questions are invited and answered by the community (e.g - GitHub, StackOverFlow (SO))

The case of open-source

Open source is good.
But every open source project is not as exquisite as linux.
One wrong merge can break down a system.
Examples :

1. Incorrect implementation of Image Cropping in CNTK (unnoticed for 2 years),
2. Incorrect implementation of atrous convolution in TensorFlow (resolved 5 months after reporting).
3. Issues with CPU vs GPU results of deconvolution in TensorFlow.

There may be some IP issues with certain components of some open source libraries.

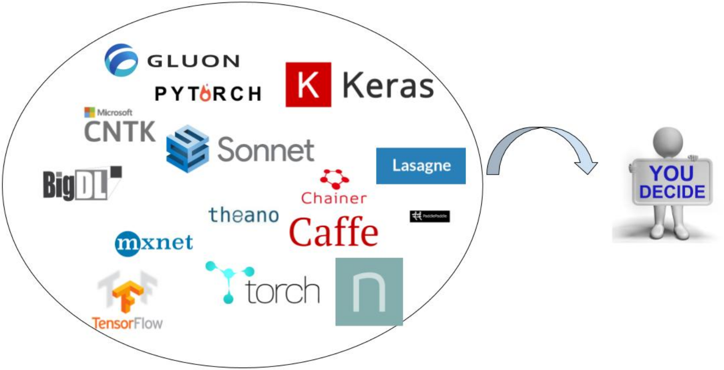
If you work in collaboration with some company(ies), be a little careful.

Examples

1. Microsoft has restricted usage 1-bit SGD algorithm included in CNTK.
2. Google holds a patent on Dropout technique.
3. A specific method of parallelizing CNN (across multiple nodes) is patented by Google.
4. A CNN based technique for data transfer in reinforcement learning is patented by Google.
5. A highly successful PCA-based augmentation of color images is patented by Google.
6. A popular approach to do conditional processing fully connected neural networks is patented by Amazon.

Selecting your library

Selection Pool for DL libraries



How do you make an informed choice ?

Our libraries of interest

- Caffe — (Berkeley Vision Lab)
- TensorFlow — (Google)
- CNTK — (Microsoft)
- Torch — (Facebook)
- PyTorch — (Facebook)
- Theano — (MILA)
- Keras — (Individual initiative + Google push)
- Lasagne — (Open source)

Libraries in **red** are not of our immediate interest.

Selection criteria for DL libraries

- Programming language support.
- Documentation quality.✓
- Community support.✓
- Learning curve.✓
- Stability.✓
- Speed
- Scalability

Criteria marked ✓ should be most vital for selection.

Selection Criteria

Programming language support

Context : What programming language(s) can you use a DL library in ?

	Python	C++	others
Caffe	✓	✓	NA
TensorFlow	✓	✓	✓
Torch	NA	NA	✓
CNTK	✓	✓	NA
PyTorch	✓	NA	NA

Words of advice

1. Python is the simplest and most effective language to program in.
2. C++ support of some libraries may be little restrictive (*all functionalities aren't available.)* (e.g- TensorFlow and CNTK)

NA : Not Applicable

Documentation quality ✓

Context : How well documented is a DL library ?

	Python	C++	others	**comments**
Caffe	✓	✓	NA	Stay away
TensorFlow	✓✓✓✓	✓✓✓	✓✓✓	lack of good examples
Torch	NA	NA	✓✓✓✓	Nice but deprecating
CNTK	✓✓✓	✓✓	NA	Different terminology
PyTorch	✓✓✓✓	NA	NA	Probably the best

Words of advice

1. Project success is proportional to documentation quality.
2. DL libraries get updated fast. Keep a perpetual eye.

NA : Not Applicable

Community support ✓

Context : How much help can you expect from other users of a DL library ?

- Community support is generally fair across all libraries.
- TensorFlow is a little exception.
  - It is tore between GitHub and SO and for deep questions disappointment is the most common reward.
- PyTorch and CNTK deserve special mention.
  - Community support is typically exceptionally good.

Words of advice

1. Community support is extremely crucial to quickly resolve issues and difficulties.
2. It is highly recommended to avoid any library with poor community support.
3. It is recommended to scrounge SO and GitHub to gauge the community enthusiasm.

Learning pace ✓

Context : How fast can you start using a DL library ?

	Python	C++	others
Caffe	✓✓✓	✓	NA
TensorFlow	✓✓	✓	NA
Torch	NA	NA	✓✓✓
CNTK	✓✓✓✓	✓	NA
PyTorch	✓✓✓✓	NA	NA

1. Avoid libraries with slow learning pace for crucial projects.

Stability ✓

Context

1. How difficult is installing a library for you ?
2. Do you have to ship the library for every DL project you work on ? (e.g- FRCNN, SSD, RPN+BT)

	Stability
Caffe	✓
TensorFlow	✓✓✓✓
Torch	✓✓✓✓
CNTK	✓✓✓✓
PyTorch	✓✓✓✓

1. Libraries with poor stability reduce your project's reliability.
2. Poor stability also impedes your work's acceptance in the community.

Speed

Context : How good is the processing speed of a DL library ?

- All the libraries we have considered are reasonably speedy.
- CNTK is exceptionally fast with RNNs and LSTMs.

1. In research speed is a secondary factor in DL.
2. Speed is **HIGHLY** contextual.
   - Your choice of a network affects the speed.
   - Your choice of network output(s) affects the speed.
   - The current state of a GPU (not just the model !!) such as temperature may also affect the speed.

Scalability

Context : Can you use a DL library across multiple GPUs on a single machine (multi-GPU setup) or distributed across multiple machines (distributed setup) ?

	Stability	Comments	Ease of use
Caffe	✓✓✓	Limited to 8 GPUs.	Very easy
TensorFlow	✓✓✓✓	**Supports arbitrarily distributed GPUs** with no upper limit.	Very hard
Torch	✓✓✓✓	Previously limited to 8 GPUs. **Now supports distributed GPUs.**	Easy
CNTK	✓✓✓✓	**Supports arbitrarily distributed GPUs** with no upper limit.	Very easy
PyTorch	✓✓✓	**Supports arbitrarily distributed GPUs** with no upper limit.	Easy

Summary of criteria

Criterion specific choices

	My choice
Language support	PyTorch
Documentation	PyTorch
Community support	CNTK
Learning Curve	CNTK
Speed	CNTK
Stability	TensorFlow
Scalability	CNTK

Holistic choice (considering a balance of all the criteria)

PyTorch

1. These choices are biased for I have used all the aforementioned libraries.

Getting comfortable with a library

Before you make a selection- I

- Clarify in your mind
  1. The kind of ideas you'd like to implement.
     - Any special data augmentation, set layer-wise learning rate, some unconventional network implementation ?
     - Joint training of a network and a classifier, dimensionality reduction ?
     - Implement some new activation function, some new initialization scheme ?
  2. The kind of analyses you'd like to do.
     - Extract layer-wise features, layer-wise performance analysis, substitute different types of classifiers ?
  3. The kind of resources you might need (e.g - multi-GPU, distributed etc.)
     - Can you work with only one GPU ?
     - Do you have a network which does not fit in one GPU ?
     - In such an event, does your DL library allow for network splitting across GPUs ?

Before you make a selection -II

- Go through the library documentation and check if they satisfy your requirements.
- If they do, make a rough summary of how your ideas can be implemented.

Always remember

1. Unless it is very convenient, don't overuse codes written by others.
2. Others' codes might make life tough at a later stage.
3. A few days of work done in library selection can save many weeks in advance.

Once you want to stay long-term with a library

- Write a code for training a large network on a large dataset from scratch.
  1. This takes you through a tour of much of a library allowing you to know it better.
  2. This is probably the **BEST** way to learn a library.
- At least take a comprehensive code and try to read and understand it.
  1. FRCNN (for object detection)
  2. T-CNN (for object tracking)
  3. Codes for some comprehensive paper in your research area.
- Understand finer subtleties of libraries.
  1. Do they expect BGR or RGB ordered images ? Do they perform any rescaling on images ?
- Derive minimum inspiration from online chatter about DL libraries.

Vital random musings

Keep these close to mind -I

- In practice multiple versions of a model are trained and the best one is selected.
  1. This is to counter randomness because SGD is stochastic. (**Almost**) all things the same, you will never get the same optima twice.
  2. Never train a network from scratch for some serious project work.
- Avoid converting trained model(s) from one library format to other.(*may cause precision loss or big errors due to implementation error(s)*
- Make sure you know which GPU model you are compiling the library for.(*FRCNN compiled on Titan X (Maxwell) will give wrong bounding boxes for GTX 1080 (Pascal)*)
- Best practice is to compile a library from sources and avoid repository install. This may affect speed significantly (*TensorFlow is about 2x slower if not compiled from sources*).
- Keep an eye on the community to ensure you resolve any bugs.

Keep these close to mind -II

- DL libraries are continuously evolving.
  1. Closely follow libraries' development.
  2. Try to read libraries' source code (esp. if thou art a PhD student)
     - 3 CNTK bugs were resolved simply by reading the source code and reporting them.
     - CNTK and PyTorch source codes are excellently written and maintained.
- Release your DL code if there has to be a long-term value of it.
  1. OverFeat (a NIPS oral paper + ILSVRC2013 winner + only paper to publish results on ILSVRC detection) is not discussed today (*No source code*)
  2. SPP-Net (A TPAMI paper of 2015) is rarely mentioned in even related work (*No proper source code*)

The isssue of Datasets

- Most of the time in DL training/fine-tuning elapses in data reading.
- Minimize the time a GPU is idle. While GPU processes process data, make sure data is transferred from CPU to GPU.
  1. All libraries offer functionalities to do so. You'd need to explore though.
  2. In training InceptionV3 (from scratch) on TensorFlow, I managed a speedup from 64 images/sec (training on 1 GPU) to 430 images/sec, after using an optimized approach of data handling.
- Always study a dataset before using. It is the most common source of errors.
- Create a general design pattern (e.g - Abstract base classes) to handle multiple datasets. This leads to fewer or no code modifications when dealing with multiple datasets.

Ensuring that your code is reproducible

- Try to set a fixed seed if you are training a model.
  - A seed is an integer determining the exact values of random numbers across systems.
  - A fixed seed (e.g - 10) ensures that others training/finetuning your system will always get the same results.
- Run your code on a different machine and check your results.
- If you use multi-GPU/distributed computing, always check it on single GPU.
- Ship the Git hash of your library.
  - A Git Hash uniquely identifies the exact version of your library.

Why I did not discuss the following

- **Theano** - Active development has ceased.
- **Keras, Lasagne, TF-Learn** - These are built on top of other libraries to make them even easier to program. But this can get very serious
    1. Usually these *add-on* are unable to offer full functionality of their base libraries.
    2. Usually these are behind their base libraries in terms of functionality.
    3. Usually these are more unstable.

These libraries are suitable for reimplementing well-established ideas.

A humble suggestion

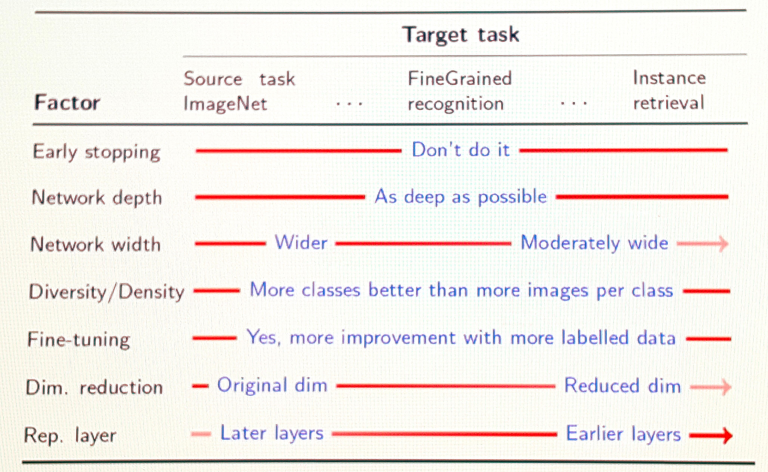
Please avoid these libraries in original research (for your own good !!!)

Fine-Tuning

Let's discuss fine-tuning

- Fine-tuning means taking a previously trained network and modifying it by training it partially (only some layers)
  1. If the dataset or task remains the same, we generally simply call it *fine-tuning*.
  2. If the task is different but somewhat related to the original task, we call it *transfer learning* or *domain transfer*.
- For implementation purposes keep in mind -
  1. RGB/BGR channel order used during original training ?
  2. Was there any pre-processing such as normalization ?
  3. **Fine-tune** over large epochs. 5-8 epochs work just fine (*even with ResNet-152*).
  4. Keep in mind the difference between the original task for which the network was trained and the task for which the fine-tuning is being done.

Fine-Tuning Wisdom



	Target task				
Factor	Source task ImageNet	· · ·	FineGrained recognition	· · ·	Instance retrieval
Early stopping			Don't do it		
Network depth			As deep as possible		
Network width	Wider				Moderately wide
Diversity/Density			More classes better than more images per class		
Fine-tuning			Yes, more improvement with more labelled data		
Dim. reduction	Original dim				Reduced dim
Rep. layer	Later layers				Earlier layers

Taken from Azizpour et.al , **"Factors of Transferability for a Generic ConvNet Representation"**, TPAMI 2015

In the end

- DL libraries should be carefully selected.
- Implementation issues in DL research should be acknowledged and discussed.
- A good selection should be
  1. Centered around project requirements.
  2. Mindful of different aspects of DL based system implementation.
- A good researcher should follow libraries' development perpetually.
- Get skilled in more than one library.
- Working fast is very bad in DL. Work slowly and gradually — A stitch in time loses nine when you do DL.