Lessons Learned from Deploying Deep Learning at Scale

O’Reilly AI 2016
September 27, 2016

Kenny Daniel
CTO @ Algorithmia
Who Am I?

Kenny Daniel - CTO, Algorithmia

- Graduate research in Artificial Intelligence and Mechanism Design
- Multiple published algorithms and papers in Machine Learning
- Received $1 million from DOT “Engineering Tomorrow’s Transportation Market”
- B.S. Carnegie Mellon University, M.S., Ph.D. (on leave) USC
- Data Scientist and Computer Vision specialist for Delectable
- Co-founder and architect of Algorithmia Platform
Algorithmia is a fundamentally new approach to AI:

• First we are making AI available to any developer on earth, no longer limited to tech giants and research labs.

• Our unique approach to acquiring algorithms allows us to provide the broadest selection of live, next-generation intelligence micro-services, shareable across applications and teams.

• The collection of these “AI modules” under one roof makes it the ideal breeding ground for developers dedicated to creating the next generation of AI.
Recently launched support for Deep Learning on Algorithmia

http://blog.algorithmia.com/2016/07/cloud-hosted-deep-learning-models
Hosting deep learning

Cloud hosting of deep learning models can be especially challenging due to complex hardware and software dependencies.

This talk will cover:

- GPU computing is essential to deep learning, but not yet mainstream
- Comparison of deep learning frameworks
- What are the challenges faced when hosting deep learning models
What is Deep Learning?

• Deep Learning uses artificial neurons similar to the brain to represent high-dimensional data
  • The mammal brain is organized in a deep architecture, e.g. visual system has 5 to 10 levels. [1]

• Deep learning excels in tasks where the basic unit (a single pixel, frequency, or word) has very little meaning in and of itself, but contains high level structure. Deep nets have been effective at learning this structure without human intervention.

Applications

Primarily: huge growth in unstructured data

- Pictures
- Videos
- Audio
- Speech
- Websites
- Emails
- Reviews
- Log files
- Social Media
Commercial Applications

- Computer Vision
  - Image classification
  - Object detection
  - Face recognition

- Natural Language
  - Speech to text
  - Chatbots
  - Q&A systems (siri, alexa, google now)
  - Machine translation

- Optimization

- Anomaly Detection

- Recommender systems
Why Now?

... and why is Deep Learning suddenly everywhere?

Advances in research

• LeCun, *Gradient-Based Learning Applied to Document Recognition*, 1998

Advances in hardware

• GPUs: 10x performance, 5x energy efficiency

http://www.nvidia.com/content/events/geoInt2015/LBrown_DL_Image_ClassificationGEOINT.pdf
Deep Learning Hardware (2016)

GPUs: Nvidia is dominating

One of the first GPU neural nets was on a NVIDIA GTX 280 up to 9 layers neural network. (2010 Ciresan and Schmidhuber)

• Nvidia chips tend to outperform AMD
• More importantly, all the major frameworks use CUDA as first-class citizen. Poor support for AMD’s OpenCL
Deep Learning Hardware

GPU
• Becoming more tailored for deep learning (eg- Pascal)

Custom hardware
• FPGA
  • MS Project Catapult
• ASIC
  • Google TPU
  • IBM TrueNorth
  • Nervana Engine
GPU Deep Learning Dependencies

- Meta deep learning framework
- Deep learning framework
- cuDNN
- CUDA
- Nvidia driver
- GPU
Deep Learning Frameworks

TensorFlow, Caffe, Torch, Theano
MXnet, DL4J, CNTK, Keras
Theano

Created by Université de Montréal

Theano pioneered the trend of using symbolic graph for programming a network. Very mature framework, good support for many kinds of networks.

Pros:
- Uses Python + Numpy
- Declarative computational graph
- Good support for RNNs
- Wrapper frameworks make it more accessible (Keras, Lasagne, Blocks)
- BSD License

Cons:
- Low level framework
- Error messages can be unhelpful
- Large models can have long compile times
- Weak support for pretrained models
Torch

Created by a collaboration of various researchers. Used by DeepMind (prior to google) Torch is a general scientific computing framework for lua.

Torch is more flexible than TensorFlow and Theano in that it is imperative while TF/Theano are declarative. That makes some operations, e.g. beam search, much easier to do in Torch.

Pros:
- Very flexible multidimensional array engine
- Multiple backends (CUDA and OpenMP)
- Lots of pre-trained models available

Cons:
- Lua
- Not good for recurrent networks
- Lack of commercial support
Caffe

Created by Berkeley Vision and Learning Center and community contributors

Probably the most used framework today, certainly for CV

Pros:
- Optimized for feedforward networks, convolutional nets, and image processing
- Simple python API
- BSD License

Cons:
- C++ / CUDA for new GPU layers
- Limited support for recurrent networks (recently added)
- Cumbersome for big networks (GoogLeNet, ResNet)
TensorFlow

Created by Google

TensorFlow is written with a Python API over a C/C++ engine. TensorFlow generates a computational graph (e.g. a series of matrix operations) and performs automatic differentiation.

Pros:
- Uses Python + Numpy
- Lots of interest from the community
- Highly parallel, and designed to use various backends (software, gpu, asic)
- Apache License

Cons:
- Slower than other frameworks
- More features, more abstractions than torch
- Not many pretrained models yet

Networks for Training

Where to get networks:
• If you’re just interested in using deep learning to classify images, you can usually find off-the-shelf networks
• VGG, GoogleNet, AlexNet, SqueezeNet
• Caffe Model Zoo
Training vs Running

Deep Learning generally consists of two phases: training and running.

Training deep learning models is challenging, with many solutions available today.

Running deep learning models is the next step, and has its own challenges.
Hosting Deep Learning

Making deep learning models available as an API presents a unique set of challenges that are rarely, if ever, addressed in the tutorials.
Why ML in the Cloud?

- Need to react to live user data
- Don’t want to manage own servers
- Need enough servers to sustain max load. Can save money using cloud services.
- Limited compute capacity on mobile
Service Oriented Architecture

Going to want dedicated infrastructure for handling computationally intensive tasks like deep learning.
Challenge #1: Infrastructure Providers

**AWS**: g2.2xlarge instances have GRID K520 cards since Nov. 2013

**Azure**: no GPU compute

**Google**: no GPU compute

**SoftLayer**: various cards including Tesla K80 & M60

**Smaller providers**: Nimbix, Cirrascale, Penguin

(Sept 2016)
Challenge #2: Language bindings

You probably already have an existing stack in some programming language. How does it talk to deep learning framework?

• Hope you like python (or lua)

Solution: services!
Challenge #3: Large Models

Deep learning models are getting larger

- State-of-the-art networks are easily multi-gigabyte
- Need to be loaded and scaled

Solutions:

- More hardware
- Smaller models

```
$ ./deep_learn.sh
exited with code -99
$ 
```
## Memory per model

<table>
<thead>
<tr>
<th>Model</th>
<th>Size (mb)</th>
<th>Error % (top-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SqueezeNet Compressed</td>
<td>0.6</td>
<td>19.7%</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>4.8</td>
<td>19.7%</td>
</tr>
<tr>
<td>AlexNet</td>
<td>240</td>
<td>19.7%</td>
</tr>
<tr>
<td>Inception v3</td>
<td>84</td>
<td>5.6%</td>
</tr>
<tr>
<td>VGG-19</td>
<td>574</td>
<td>7.5%</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>102</td>
<td>7.8%</td>
</tr>
<tr>
<td>ResNet-200</td>
<td>519</td>
<td>4.8%</td>
</tr>
</tbody>
</table>
SqueezeNet

Hypothesis: the networks we’re using today are much larger and more complicated than they need to be.

Enter SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size

Not quite state-of-the-art, but close, and MUCH easier to host.

[1] Iandola, Han, Moskewicz, Ashraf, Dally, Keutzer (2016)
Model Compression

Recent efforts aimed at pruning the size of networks

“Reduce the storage requirement of neural networks by 35x to 49x without affecting their accuracy.”

(Han, Mao, Dally - 2015)
Challenge #4: GPU Sharing

GPUs were not designed to be shared like CPUs

- Limited amount of video memory
- Even with multi-context mgmt, memory overflows and unrestricted pointer logic are very dangerous for other applications
- Devs need a way to share GPU resources safely from potentially malicious applications
Challenge #5: GPU Sharing - Docker

Docker - new standard in deploying applications, but adds an additional layer of challenges to GPU computing.

• Nvidia drivers must match inside and outside containers
• CUDA drivers must match inside and outside containers
• Some algorithms require X windows, which must be started outside the docker container and mounted inside
• nvidia-docker container is helpful but not a complete solution
Lessons Learned

• Deep learning in the cloud is still in its infancy
• Hosting deep learning models is the next logical step after training model, but the difficulty is under-appreciated
• Tooling and frameworks are making things easier, but a lot of opportunity for improvement

Big picture: the challenges involved with creating DL models is only half the problem. Deploying them is an entirely different skillset.
ALGORITHMIA
algorithmia.com/signup?invite=OReillyAI2016

Thank you!

Kenny Daniel
CTO

kenny@algorithmia.com
@platypii