ANIMA ANANDKUMAR
DISTRIBUTED DEEP LEARNING
PRACTICAL CONSIDERATIONS FOR MACHINE LEARNING

SOFTWARE PACKAGES

UTILITY COMPUTING

DIVERSE & LARGE DATASETS
CHALLENGES IN DEPLOYING LARGE-SCALE LEARNING
CHALLENGES IN DEPLOYING LARGE-SCALE LEARNING

• Complex deep network
  • Coding from scratch is impossible
• A single image requires billions floating-point operations
  • Intel i7 ~500 GFLOPS
  • Nvidia Titan X: ~5 TFLOPS
• Memory consumption is linear with number of layers
DESIRABLE ATTRIBUTES IN A ML SOFTWARE PACKAGE

PROGRAMMABILITY
Simplifying network definitions

PORTABILITY
Efficient use of memory

EFFICIENCY
In training and inference
mxnet
theano
TensorFlow
Caffe
torch
CNTK
MXNET IS AWS’ S DEEP LEARNING FRAMEWORK OF CHOICE
MOST OPEN
Apache

BEST ON AWS
(Integration with AWS)
single implementation of backend system and common operators

performance guarantee regardless which front-end language is used
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1

**IMPERATIVE PROGRAMMING**

**Easy to tweak with python codes**

**PROS**
- Straightforward and flexible.
- Take advantage of language native features (loop, condition, debugger)
- E.g. Numpy, Matlab, Torch, ...

**CONS**
- Hard to optimize
A = Variable('A')
B = Variable('B')
C = B * A
D = C + 1
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)

**PROS**
- More chances for optimization
- Cross different languages
- E.g. TensorFlow, Theano, Caffe

**CONS**
- Less flexible

*C can share memory with D because C is deleted later*
**MXNET: MIXED PROGRAMMING PARADIGM**

**IMPERATIVE NDARRAY API**

```python
>>> import mxnet as mx
>>> a = mx.nd.zeros((100, 50))
>>> b = mx.nd.ones((100, 50))
>>> c = a + b
>>> c += 1
>>> print(c)
```

**DECLARATIVE SYMBOLIC EXECUTOR**

```python
>>> import mxnet as mx
>>> net = mx.symbol.Variable('data')
>>> net = mx.symbol.FullyConnected(data=net, num_hidden=128)
>>> net = mx.symbol.SoftmaxOutput(data=net)
>>> texec = mx.module.Module(net)
>>> texec.forward(data=c)
>>> texec.backward()
```

NDArray can be set as input to the graph
MXNET: MIXED PROGRAMMING PARADIGM

```python
texec = mx.module.Module(net)
for batch in train_data:
    texec.forward(batch)
    texec.backward()
    for param, grad in zip(texec.get_params(), texec.get_grads()):
        param -= 0.2 * grad
```

Embed symbolic expressions into imperative programming
PORTABILITY
• Fit the core library with all dependencies into a single C++ source file
• easy to compile on any platform
MEMORY OPTIMIZATION

TRADEOFF MEMORY FOR COMPUTATION

- Needs an extra forward pass
- Reduces the memory complexity from $O(n)$ to $O(\sqrt{n})$, where $n$ is the number of layers
### EXAMPLES

<table>
<thead>
<tr>
<th>Model</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>130 GB</td>
<td>4 GB</td>
</tr>
<tr>
<td>LSTM</td>
<td>270 GB</td>
<td>2.5 GB</td>
</tr>
</tbody>
</table>

- **ResNet**
  - 1000 layers
  - batch size 32

- **LSTM**
  - 4 layers
  - 1000 hidden size
  - 1000 unroll
  - batch size 32
WRITING PARALLEL PROGRAMS IS HARD

Dependency graph for 2-layer neural networks with 2 GPUs

Each forward-backward-update involves $O(\text{num\_layer})$, which is often 100–1,000, tensor computations and communications.
HIERARCHICAL PARAMETER SERVER IN MXNET

- Network Switch
  - 1.25 GB/s
  - 10 Gbit Ethernet
  - 15.75 GB/s
  - PCIe 3.0 16x
  - 63 GB/s
  - 4 PCIe 3.0 16x

- GPU

- PCIe Switch

- Level-2 Servers

- Level-1 Servers

- Workers
### Scalability of MXNet

<table>
<thead>
<tr>
<th>Scale to Multiple Cores</th>
<th>Scale Across GPUs</th>
<th>Scale Across Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep learning well suited to GPUs</td>
<td>Up to 16 available on P2.16xl</td>
<td>Lots and lots of p2.16xl ;</td>
</tr>
</tbody>
</table>
github.com/awslabs/deeplearning-benchmark
91% Efficiency

No. of GPUs
88% Efficiency

No. of GPUs
88% Efficiency

- Cloud formation with Deep Learning AMI
- 16x P2.16xlarge. Mounted on EFS
- Inception and Resnet: batch size 32, Alexnet: batch size 512
- ImageNet, 1.2M images, 1K classes
- 152-layer ResNet, 5.4d on 4x K80s (1.2h per epoch), 0.22 top-1 error
ROADMAP FOR MXNET

- Documentation (installation, native documents, etc.)
- Platform support (Linux, Windows, OS X, mobile …)
- Sparse datatypes and tensor operations
- Platform for general distributed machine learning algorithms
Tensors = natural representations for many data in Machine Learning (e.g. images are third order tensors (height, width, channels))

Great tool to better understand Deep Learning

Tensor decomposition has ability to discover multi-dimensional dependencies and produce compact low-rank approximation of data

Tensors are first class citizens in MxNet
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with \( 5 \times 5 \times 48 \) kernels. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size \( 3 \times 3 \times 256 \) connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size \( 3 \times 3 \times 192 \), and the fifth convolutional layer has 256 kernels of size \( 3 \times 3 \times 192 \). The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random \( 224 \times 224 \) patches (and their horizontal reflections) from the \( 256 \times 256 \) images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five \( 224 \times 224 \) patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components. This is the reason why the input images in Figure 2 are \( 224 \times 224 \times 3 \)-dimensional.

AlexNet, ImageNet classification with deep convolutional neural networks, NIPS’12, Alex Krizhevsky et. al.
TENSOR METHODS, DEEP LEARNING & MXNET

TRADITIONAL APPROACH: STRUCTURE OF THE DATA IS LOST

TENSOR METHODS: LEVERAGE THE STRUCTURE OF THE DATA
TENSORS, DEEP LEARNING & MXNET

TRADITIONAL APPROACH: STRUCTURE OF THE DATA IS LOST

TENSOR METHODS: LEVERAGE THE STRUCTURE OF THE DATA
Tucker tensor decomposition: express a tensor as a function of a low rank tensor and projection matrices

\[ \tilde{\mathcal{X}} = \tilde{\mathcal{G}} \times_1 U^{(1)} \times_2 U^{(2)} \times \cdots \times_N U^{(N)} \]
TENSOR CONTRACTION AS A LAYER

- Take activation tensor as input
- Feed it through a tensor contraction layer (TCL)
- Output a low rank activation tensor
TENSOR CONTRACTION AS A LAYER

- Compact representation
  -> less parameters
  (measured as Space Savings)

  \[
  \text{space saving} = 1 - \frac{n_{\text{original parameters}}}{n_{\text{parameters in compact model}}}
  \]

- Similar and sometimes better performance
<table>
<thead>
<tr>
<th>Method - Hidden Units in Fully Connected Layers</th>
<th>Accuracy (%)</th>
<th>Space savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Traditional AlexNet, 4096 hidden units</td>
<td>56.29</td>
<td>0</td>
</tr>
<tr>
<td>Adding a TCL (256, 5, 5), 4096 hidden units</td>
<td>57.54</td>
<td>-0.11</td>
</tr>
<tr>
<td>Adding a TCL (200, 5, 5), 3276 hidden units</td>
<td>56.11</td>
<td>35.73</td>
</tr>
<tr>
<td>Replace a FCL with (256, 5, 5) TCL, 4096 Hidden Units</td>
<td>56.63</td>
<td>44.45</td>
</tr>
</tbody>
</table>

Results on ImageNet with an AlexNet. *J. Kossafi et. al 2017*
AMIs, Cloud Formation and DL

One-Click Deep Learning
P2 INSTANCES
Up to 40k CUDA cores

DEEP AMI
Pre-configured for deep learning

DEEP TEMPLATE
Deep learning clusters
**P2 INSTANCES**

Up to 40k CUDA cores

- **p2.16xl instance** = 16 K80 GPUs ~ 70 tera flops
- 16 p2.16xl instances ~ 1.1 peta flops
- World’s fastest supercomputer ~ 93 peta flops
- GPUDirect™ (peer-to-peer GPU communication)
Amazon Machine Images


- Tool for data scientists and developers
- Setting up a DL system takes (install) time & skill
- Keep packages up to date and compiled (MXNet, TensorFlow, Caffe, Torch, Theano, Keras)
- Anaconda, Jupyter, Python 2 and 3
- NVIDIA Drivers for G2 and P2 instances
- Intel MKL Drivers for all other instances (C4, M4, …)

Deep Learning any way you want on AWS
Start with CloudFormation in Console
Introducing Amazon AI

Apache MXNet
Deep learning engine

Polly
Text-to-Speech

Rekognition
Image Analysis

Lex
ASR & NLU
Rekognition: Search & Understand Visual Content

- Real-time & batch image analysis
- Object & Scene Detection
- Facial Detection
- Facial Analysis
- Face Search
# Rekognition: Object & Scene Detection

![Beach Scene](image)

<table>
<thead>
<tr>
<th>Category</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bay</td>
<td>99.18%</td>
</tr>
<tr>
<td>Beach</td>
<td>99.18%</td>
</tr>
<tr>
<td>Coast</td>
<td>99.18%</td>
</tr>
<tr>
<td>Outdoors</td>
<td>99.18%</td>
</tr>
<tr>
<td>Sea</td>
<td>99.18%</td>
</tr>
<tr>
<td>Water</td>
<td>99.18%</td>
</tr>
<tr>
<td>Palm_tree</td>
<td>99.21%</td>
</tr>
<tr>
<td>Plant</td>
<td>99.21%</td>
</tr>
<tr>
<td>Tree</td>
<td>99.21%</td>
</tr>
<tr>
<td>Summer</td>
<td>58.3%</td>
</tr>
<tr>
<td>Landscape</td>
<td>51.84%</td>
</tr>
<tr>
<td>Nature</td>
<td>51.84%</td>
</tr>
<tr>
<td>Hotel</td>
<td>51.24%</td>
</tr>
</tbody>
</table>
Rekognition: Facial Analysis

- **Emotion**: calm: 73%
- **Sunglasses**: false (value: 0)
- **Mouth open wide**: 0% (value: 0)
- **Eye closed**: open (value: 0)
- **Glasses**: no glass (value: 0)
- **Mustache**: false (value: 0)
- **Beard**: no (value: 0)
Lex: Build Natural, Conversational Interactions In Voice & Text

Voice & Text “Chatbots”

Powers Alexa

Voice interactions on mobile, web & devices

Text interaction with Slack & Messenger (with more coming)

Enterprise Connectors
- Salesforce
- Microsoft Dynamics
- Marketo
- Zendesk
- Quickbooks
- Hubspot
“Book a flight to London”

Automatic Speech Recognition

Natural Language Understanding

“Location”

Flight Booking

Origin | Seattle
---|---
Destination | London Heathrow
Departure Date

“When would you like to fly?”

Grammar Graph

Utterances

Polly
When would you like to fly?

Flight Booking

<table>
<thead>
<tr>
<th>Origin</th>
<th>Seattle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>London Heathrow</td>
</tr>
<tr>
<td>Departure Date</td>
<td></td>
</tr>
</tbody>
</table>
Amazon Polly: Life-like Speech Service

- Converts text to life-like speech
- Fully managed
- 47 voices
- 24 languages
- Low latency, real time
Let’s listen...
Polly: A Focus On Voice Quality & Pronunciation

1. Automatic, Accurate Text Processing

“Today in Seattle, WA, it’s 11°F”

“"We live for the music" live from the Madison Square Garden.”
Polly: A Focus On Voice Quality & Pronunciation

1. Automatic, Accurate Text Processing

2. Intelligible and Easy to Understand
Polly: A Focus On Voice Quality & Pronunciation

1. Automatic, Accurate Text Processing

2. Intelligible and Easy to Understand

3. Add Semantic Meaning to Text

“Richard’s number is 2122341237“

“Richard’s number is 2122341237“

*Telephone Number*
Polly: A Focus On Voice Quality & Pronunciation

1. Automatic, Accurate Text Processing

2. Intelligible and Easy to Understand

3. Add Semantic Meaning to Text

4. Customized Pronunciation

   “My daughter’s name is Kaja.”

   “My daughter’s name is Kaja.”
• Apply for AWS credits for your research
  https://aws.amazon.com/grants/

• Apply for AWS credits for education
  https://aws.amazon.com/education/awseducate/

• Conduct research and build products at AWS:
  internships and full time positions!

• Send me an email: anima@amazon.com