Mocha.jl - Deep Learning in Julia

Chiyuan Zhang (@pluskid)
CSAIL, MIT
Overview

- Deep Learning
  - Applications (state-of-the-art)
  - Basic Introduction
- Julia
- Mocha.jl
  - IJulia Image Classification Example
  - Training a ConvNet in Mocha.jl
What is Deep Learning?

GoogLeNet (Inception)
Winner of ILSVRC 2014, 27 layers, ~7 million parameters
Why People are Excited about Deep Learning?


Why People are Excited about Deep Learning?

IBM Watson announces breakthrough in Conversational Speech Transcription

Speech recognition technology made enormous strides over the last five years. For example, it is now possible to use speech to input text into smartphones with very high accuracy. This is a critical usability feature given the difficulties that are involved in inputting text on tiny keyboards.

However, such applications give the mistaken impression that speech recognition is now a "solved problem". Nothing can be further from the truth! Speech recognition accuracy on casually spoken speech — for example, speech in conversations and meetings — is still dismally low. Even the best of the art technology from top laboratories can have difficulty getting more than 50% of the words correct in such challenging environments.

IBM Watson is proud to announce a major advance in the transcription of conversational speech. Watson researcher George Saon along with colleagues Jeff Kuo and Steve Rennie built a system capable of very low error rates on a popular scientific benchmark that consists of telephone conversations – the NIST Switchboard corpus (“EvalSet-2”). Furthermore, they achieved this by using only publicly available data (details available on request) to train the underlying models. The performance of our new system — an 8% word error rate — is 39% better than previously reported external results.

Skype realtime speech translation

Image source: Li Deng and Dong Yu. Deep Learning – Methods and Applications.
Why People are Excited about Deep Learning?

(Deep) Neural Networks

- Composition of computations (abstracted as "layers") organized as a hierarchy, or more generally directed acyclic graph (DAG).
- e.g. a fully-connected layer (a.k.a. dense layer, inner-product layer, linear layer, etc.) computes

\[ y_i = \sum_j w_{ij} x_j \]

- An activation function layer applies an entry-wise nonlinear map (e.g. sigmoid, ReLU).

\[
\begin{align*}
\text{Sigmoid } \sigma(x) &= \frac{1}{1 + e^{-x}} \\
\text{ReLU } \sigma(x) &= \max\{x, 0\}
\end{align*}
\]
To Learn More about Deep Learning

- **Online resources**
  - [http://deeplearning.net/](http://deeplearning.net/)
  - Links to more resources: [https://github.com/ChristosChristofidis/awesome-deep-learning](https://github.com/ChristosChristofidis/awesome-deep-learning)

- **Courses**
  - Neural Networks for Machine Learning [https://www.coursera.org/course/neuralnets](https://www.coursera.org/course/neuralnets)

- **Books**
  - Deep Learning: Methods and Applications [http://research.microsoft.com/apps/pubs/?id=209355](http://research.microsoft.com/apps/pubs/?id=209355)

- ...
Multiple dispatch
Dynamic type system
Good performance, approaching C
High-Performance JIT Compiler
Built-in package manager
Lisp-like macros and meta-programming
Call Python functions using the PyCall
Call C functions directly
Designed for parallelism and distributed computation
Coroutines
MIT licensed: free and open source
A Taste of Julia

```julia
function mandel(z)
    c = z
    maxiter = 80
    for n = 1:maxiter
        if abs(z) > 2
            return n-1
        end
        z = z^2 + c
    end
    return maxiter
end

function randmatstat(t)
    n = 5
    v = zeros(t)
    w = zeros(t)
    for i = 1:t
        a = randn(n,n)
        b = randn(n,n)
        c = randn(n,n)
        d = randn(n,n)
        P = [a b; c d]
        Q = [a b; c d]
        v[i] = trace((P.'*P)^4)
        w[i] = trace((Q.'*Q)^4)
    end
    std(v)/mean(v), std(w)/mean(w)
end
```

*Figure: benchmark times relative to C (smaller is better, C performance = 1.0).*
Deep Learning Toolkits / Libraries

A deep learning toolkit provides common layers, easy ways to define network architecture, and transparent interface to high performance computation backends (BLAS, GPUs, etc.)

- **C++**: Caffe (widely used on academia), dmlc/cxxnet, cuda-convnet, etc.

- **Python**: Theano (auto-differentiation) and wrappers, NervanaSystems/neon, etc.

- **Lua**: Torch7 (facebook); **Matlab**: MatConvNet (VGG)

- **Julia**: pluskid/Mocha.jl, dfdx/Boltzmann.jl
Why Mocha.jl?

• **Written in Julia and for Julia**: easily making use of data pre/post processing and visualization tools from Julia.

• **Minimum dependency**: Julia backend ready to run, easy for fast prototyping.

• **Multiple backends**: easily switching to CUDA + cuDNN based backend for highly efficient deep nets training.

• **Correctness**: all computation layers are unit-tested.

• **Modular architecture**: layers, activation functions, network topology, etc. Easily extendable.
Getting started with Mocha.jl

- Installing Julia
  - [http://julialang.org/downloads/](http://julialang.org/downloads/)
  - Download the package for your OS and unpack

- Installing Mocha.jl
  - In Julia REPL
  - Pkg.add("Mocha")
  - Or for the latest development version
    - Pkg.checkout("Mocha")
  - To run the unit test
  - Pkg.test("Mocha")
# load image for prediction
img = imread("images/cat256.jpg")

# get prediction
probs, class = classify(classifier, img)
println(class)

tabby, tabby cat
Mini-Tutor: ConvNets on MNIST

• MNIST: handwritten digits

• Data preparation:
  
  • Image data in Mocha is represented as 4D tensor: width-by-height-by-channels-by-batch
    
    • MNIST: 28-by-28-by-1-by-64
    
    • Mocha supports ND-tensor for general data
    
    • HDF5 file: general format for tensor data, also supported by numpy, Matlab, etc.
Network Starts with a Data Layer

• Data layer

```python
data_layer = AsyncHDF5DataLayer(name="train-data", source="data/train.txt", batch_size=64, shuffle=true)
```

• `data/train.txt` lists the HDF5 files for training set
• 64 images is provided for each mini-batch
• the data is shuffled to improve convergence
• async data layer use Julia’s `@async` to pre-read data while waiting for computation on CPU / GPU
conv_layer = ConvolutionLayer(name="conv1", n_filter=20, kernel=(5,5), bottoms=[:data], tops=[:conv])
pool_layer = PoolingLayer(name="pool1", kernel=(2,2), stride=(2,2), bottoms=[:conv], tops=[:pool])
Blobs & Network Architecture

- Network architecture is determined by connecting **tops** (output) blobs to **bottoms** (input) blobs with matching blob names.

- Layers are automatically sorted and connected as a directed acyclic graph (DAG).
The Rest of the Layers

conv2_layer = ConvolutionLayer(name="conv2", n_filter=50, kernel=(5,5), bottoms=[:pool], tops=[:conv2])

pool2_layer = PoolingLayer(name="pool2", kernel=(2,2), stride=(2,2), bottoms=[:conv2], tops=[:pool2])

fc1_layer = InnerProductLayer(name="ip1", output_dim=500, neuron=Neurons.ReLU(), bottoms=[:pool2], tops=[:ip1])

fc2_layer = InnerProductLayer(name="ip2", output_dim=10, bottoms=[:ip1], tops=[:ip2])

loss_layer = SoftmaxLossLayer(name="loss", bottoms=[:ip2, :label])
Stochastic Gradient Descent Solver

```python
params = SolverParameters(max_iter=10000,
                      regu_coef=0.0005,
                      mom_policy=MomPolicy.Fixed(0.9),
                      lr_policy=LRPolicy.Inv(0.01, 0.0001, 0.75),
                      load_from=exp_dir)

solver = SGD(params)
```
Coffee Breaks...

for the solver

setup_coffee_lounge(solver, save_into="$exp_dir/statistics.jld", every_n_iter=1000)

# report training progress every 100 iterations
add_coffee_break(solver, TrainingSummary(), every_n_iter=100)

# save snapshots every 5000 iterations
add_coffee_break(solver, Snapshot(exp_dir), every_n_iter=5000)
Solver Statistics

- Solver statistics will be automatically saved if coffee lounge is set up.

- Snapshots save the training progress periodically, can continue training from the last snapshot after interruption.
Demo: CPU vs GPU

```
backend = use_gpu ? GPUBackend() : CPUBackend()
```
Parameter Sharing

• When a layer has trainable parameters (e.g. convolution, inner-product layers), those parameters will be registered under the layer name, and shared by layers with the same name

• Use cases
  • Validation network during training
  • Pre-training, fine-tuning
  • Advanced architectures, time-delayed nodes
Parameter Sharing
Thank You!