Lecture 2:

Caffe: getting started

Forward propagation

boris.ginzburg@intel.com
Agenda

• Caffe – getting started
• Test description
• Network topology definition
• Basic layers: definition and forward propagation
  – Convolutional
  – Pooling
  – ReLU
  – Fully Connected layer
  – Softmax
• Implementation details of Convolutional layer
• MNIST training
Open-source Deep Learning libraries

   Very fast. C++/ CUDA, Python and Matlab wrappers

2. [https://code.google.com/p/cuda-convnet2/](https://code.google.com/p/cuda-convnet2/)
   Just released. Excellent tutorial. Best Cuda code.

   Excellent tutorial, C++/Cuda, Lua.

4. [http://deeplearning.net/software/pylearn2/](http://deeplearning.net/software/pylearn2/)
   Integrated with Theano, C++/Cuda, Python

   C++/CUDA.
Caffe:
Convolutional Architecture for Fast Feature Embedding

Created by Yangqing Jia
Developed by BVLC
caffe.berkeleyvision.org
bvlc.eecs.berkeley.edu
Caffe: installation

1. Ubuntu 12.04
2. Cuda 5.5 or 6.0
   - SDK - required, NVidia card is optional 😊
3. BLAS:
   - OpenBLAS or Intel MKL(Math Kernel Lib)

$ git clone https://github.com/BVLC/caffe
Caffe: example 1 - MNIST

Caffe: database format

src/tools/convert_mnist_data.cpp: MNIST format → leveldb

1. leveldb: https://code.google.com/p/leveldb/
   - <key,value>: arbitrary byte arrays; data is stored sorted by key; callers can provide a custom comparison function to override the sort order.
   - basic operations: Put(key,value), Get(key), Delete(key).

2. caffe “dev” branch supports lmdb: http://symas.com/mdb/
   - <key;value>; data is stored sorted by key
   - uses memory-mapped files: the read performance of a pure in-memory db while still offering the persistence of standard disk-based db
   - concurrent
Caffe: configuration files

1. Solver descriptor:

1. Net descriptor:
   - http://caffe.berkeleyvision.org/mnist_prototxt.html

Parameters are defined in src/caffe/proto/caffe.proto.

Protobuf (Google protocol buffers) format - easy-to-use automatic generation of configuration files:
https://developers.google.com/protocol-buffers/docs/overview
LeNet Topology

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet topology

FORWARD

Soft Max

Inner Product

ReLUP

Inner Product

Pooling [2x2, stride 2]

Convolutional layer [5x5]

Pooling [2x2, stride 2]

Convolutional layer [5x5]

Data Layer

BACKWARD
Layer:: Forward( )

class Layer {
    Setup (bottom, top);       // initialize layer
    Forward (bottom, top);    // compute next layer
    Backward( top, bottom);   // compute gradient
}

Layer:: Forward( ) propagate $y_{l-1}$ to next layer:

$$y_l = f(w_l, y_{l-1})$$
Data Layer

name: "mnist"
type: DATA
data_param {
    source: "mnist-train-leveldb"
    batch_size: 64
    scale: 0.00390625
}
top: "data"
top: "label"
Convolutional Layer

name: "conv1"
type: CONVOLUTION
blobs_lr: 1.
convolution_param {
  num_output: 20
  kernelsize: 5
  stride: 1
  weight_filler {
    type: "xavier"
  }
  bias_filler {
    type: "constant"
  }
}
bottom: "data"
top: "conv1"
Convolutional Layer

for (n = 0; n < N; n++)
  for (m = 0; m < M; m++)
    for(y = 0; y < Y; y++)
      for(x = 0; x < X; x++)
        for(p = 0; p < K; p++)
          for(q = 0; q < K; q++)
            \[ y_L(n; x, y) \] += \[ y_{L-1}(m, x+p, y+q) \] * w (m, n; p, q);

Add bias...
Pooling Layer

name: "pool1"
type: POOLING
pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
}
bottom: "conv1"
top: "pool1"

for (p = 0; p< k; p++)
    for (q = 0; q< k; q++)
        \[ y_L(x, y) = \max ( y_L(x, y), y_{L-1}(x*s + p, y*s + q) ) \];

Pooling helps to extract features that are increasingly invariant to local transformations of the input image.
 Inner product (Fully Connected) Layer

\[ Y_L(n) = \sum W_L(n, m) \ast Y_{L-1}(m) \]

name: "ip1"
  type: INNER_PRODUCT
  blobs_lr: 1.
  blobs_lr: 2.
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "pool2"
  top: "ip1"
ReLU Layer

layers {
    name: "relu1"
    type: RELU
    bottom: "ip1"
    top: "ip1"
}

\[ Y_L(n; x, y) = \max( Y_{L-1}(n; x, y), 0 ) ; \]
SoftMax + Loss Layer

Combines softmax:

\[ Y_L[i] = \frac{\exp(Y_{L-1}[i])}{\sum(Y_L[i])} \]

with log-loss:

\[ E = - \log(Y_L(label(n))) \]
LeNet topology

- **Data Layer**: 1x28x28
- **Convolutional layer [5x5]**: 20x24x24
- **Pooling [2x2, stride 2]**: 20x12x12
- **Convolutional layer [5x5]**: 50x8x8
- **Pooling [2x2, stride 2]**: 50x4x4
- **Inner Product**: 500x1
- **Inner Product**: 10x1
- **ReLUP**: 500x1
- **Soft Max**: 10x1
SOME IMPLEMENTATION DETAILS
Data Layer

All data is stored as **BLOBs - Binary (Basic) Large Objects**

```cpp
class Blob {
    Blob(int num, int channels, int height, int width);
    const Dtype* cpu_data() const;
    const Dtype* gpu_data() const;
    ...

    protected:
    shared_ptr<SyncedMemory> data_; // container for cpu_ / gpu_memory
    shared_ptr<SyncedMemory> diff_; // gradient
    int num_;  
    int channels_;  
    int height_;  
    int width_;  
    int count_;  
```
Conv Layer implementation is based on reduction to matrix-matrix multiply (See Chellapilla et al., “High Performance Convolutional Neural Networks for Document Processing”)

Figure 2. Example convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted). The top figure presents the traditional convolution operations, while the bottom figure presents the matrix version.
Convolutional Layer: im2col

\[ O_x = ((l_x - K_x + 1) + (S_x - 1))/S_x \quad O_x: \text{output width, } S_x: \text{horizontal subsampling} \]

\[ O_y = ((l_y - K_y + 1) + (S_y - 1))/S_y \quad O_y: \text{output height, } S_y: \text{vertical subsampling} \]

Figure 3. Unrolling the convolution operations in a convolutional layer (biases, sub-sampling, and non-linearity omitted), to produce a matrix-matrix product version.
Convolutional Layer: groups

AlexNet topology (Imagenet)

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.
Exercises

1. Play with Mnist topologies
   - How does the net accuracy depend on topology?

2. Port one of datasets http://deeplearning.net/datasets:
   - NORB, SVHN, ...

3. Look at the definition of following layers:
   - sigmoid, tanh, ...