Caffe tutorial

borrowed slides from:
caffe official tutorials
Recap Convnet

Supervised learning trained by stochastic gradient descend

\[ J(W, b) = \frac{1}{2} \| h(x) - y \|^2 \]

1. feedforward: get the activations for each layer and the cost
2. backward: get the gradient for all the parameters
3. update: gradient descend
Outline

- For people who use CNN as a blackbox
- For people who want to define new layers & cost functions
- A few training tricks.

* there is a major update for caffe recently, we might get different versions
Blackbox Users

http://caffe.berkeleyvision.org/tutorial/
highly recommended!
Installation

detailed documentation:
http://caffe.berkeleyvision.org/installation.html

required packages:

• **CUDA, OPENCV**

• **BLAS** (Basic Linear Algebra Subprograms): 
  *operations like matrix multiplication, matrix addition, both implementation for CPU(cBLAS) and GPU(cuBLAS). provided by MKL(INTEL), ATLAS, openBLAS, etc.*

• **Boost**: a c++ library.
  > *Use some of its math functions and shared_pointer.*

• **glog, gflags** provide logging & command line utilities.
  > *Essential for debugging.*

• **leveldb, lmdb**: database io for your program.
  > *Need to know this for preparing your own data.*

• **protobuf**: an efficient and flexible way to define data structure.
  > *Need to know this for defining new layers.*
Preparing data

—> If you want to run CNN on other dataset:

• caffe reads data in a standard database format.

• You have to convert your data to leveldb/lmdb manually.

```markdown
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  # the DATA layer configuration
  data_param {
    # path to the DB
    source: "examples/mnist/mnist_train_lmdb"
    # type of DB: LEVELDB or LMDB (LMDB supports concurrent reads)
    backend: LMDB
    # batch processing improves efficiency.
    batch_size: 64
  }
  # common data transformations
  transform_param {
    # feature scaling coefficient: this maps the [0, 255] MNIST data to
  }
```
Preparing data

this is the only coding needed (chenyi has experience)

```cpp
// declare databases
MDB_env *mdb_env;
MDB_dbi mdb_dbi;
MDB_val mdb_key, mdb_data;
MDB_txn *mdb_txn;
// leveldb
leveldb::DB* db;

// Open db
if (db_backend == "leveldb") { // leveldb
    LOG(INFO) << "Opening leveldb " << db_path;
    leveldb::Status status = leveldb::DB::Open(
        options, db_path, &db);
} else if (db_backend == "lmdb") { // lmdb
    LOG(INFO) << "Opening lmdb " << db_path;
    CHECK_EQ(mdb_open(mdb_txn, NULL, 0, &mdb_dbi), MDB_SUCCESS) << "mdb_open failed. Does the lmdb already exist? ";
}

// Initialize the leveldb
leveldb::DB* db_temp;
db->reset(db_temp);
iter.reset(db_temp->NewIterator(leveldb::ReadOptions()));
iter.SeekToFirst();
// Read a data point, and use it to initialize the top blob.
Datum datum;
datum.ParseFromString(iter.value().ToString());

// how caffe loads data in data_layer.cpp
// (you don’t have to know)
```

example from mnist: `examples/mnist/convert_mnist_data.cpp`
define your network

→ If you want to define your own architecture

net:
  **blue**: layers you need to define
  **yellow**: data blobs

  - **name**: "dummy-net"
  - **layers** { name: "data" ... }
  - **layers** { name: "conv" ... }
  - **layers** { name: "pool" ... }
    - ... more layers ...
  - **layers** { name: "loss" ... }

examples/mnist/lenet_train.prototxt
define your network

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

name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
convolution_param {
  num_output: 20
  kernel_size: 5
  stride: 1
  weight_filler {
    type: "xavier"
  }
}

examples/mnist/lenet_train.prototxt
```
define your network

loss:

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
define your network

→ a little more about the network

• network does not need to be linear

linear network:

![Linear Network Diagram](image)

directed acyclic graph:

![Directed Acyclic Graph](image)
define your solver

- solver is for setting training parameters.

```plaintext
train_net: "lenet_train.prototxt"
base_lr: 0.01
lr_policy: "constant"
momentum: 0.9
weight_decay: 0.0005
max_iter: 10000
snapshot_prefix: "lenet_snapshot"
solver_mode: GPU

examples/mnist/lenet_solver.prototxt
```
train your model

-> you can now train your model by

```
./train_lenet.sh
```

```
TOOLS=../../build/tools
GLOG_logtostderr=1 $TOOLS/train_net.bin lenet_solver.prototxt
```
finetuning models

—> what if you want to transfer the weight of a existing model to finetune another dataset / task

• Simply change a few lines in the layer definition

```
layers {  
    name: "data"  
    type: DATA  
    data_param {  
        source:  
            "ilsvrc12_train_leveldb"  
        mean_file: "../../data/ilsvrc12"  
    }  
}  

layers {  
    name: "data"  
    type: DATA  
    data_param {  
        source:  
            "style_leveldb"  
        mean_file: "../../data/ilsvrc12"  
    }  
}  
```

new name = new params

```
layers {  
    name: "fc8"  
    type: INNER_PRODUCT  
    blobs_lr: 1  
    weight_decay: 1  
    weight_decay: 0  
    inner_product_param {  
        num_output: 1000  
    }  
}  

layers {  
    name: "fc8-style"  
    type: INNER_PRODUCT  
    blobs_lr: 1  
    weight_decay: 1  
    weight_decay: 0  
    inner_product_param {  
        num_output: 20  
    }  
}  
```
finetuning models

old caffe:

```bash
> finetune_net.bin solver.prototxt model_file
```

new caffe:

```bash
> caffe train -solver models/finetune_flickr_style/solver.prototxt
   -weights bvlc_reference_caffenet.caffemodel
```

Under the hood (loosely speaking):
```cpp
net = new Caffe::Net("style_solver.prototxt");
net.CopyTrainedNetFrom(pretrained_model);
solver.Solve(net);
```
extracting features

Run:

```
build/tools/extract_features.bin imagenet_model imagenet_val.prototxt fc7 temp/features 10
```

- **network definition**
- **data blobs you want to extract**
- **output_file**
- **model_file**
- **batch_size**
MATLAB wrappers

-> What about importing the model into Matlab memory?

install the wrapper:

> make matcaffe

• RCNN provides a function for this:

> model = rcnn_load_model(model_file, use_gpu);

https://github.com/rbgirshick/rcnn
More curious Users
nsight IDE

-> needs an environment to program caffe? use nsight

• nsight automatically comes with CUDA, in the terminal hit “nsight”

For this nsight eclipse edition, it supports nearly all we need:

• an editor with highlight and function switches
• debug c++ code and CUDA code
• profile your code
Protobuf

- understanding protobuf is very important to develop your own code on caffe
- protobuf is used to define data structure for multiple programming languages

```protobuf
define message student {
    string name = 3;
    int ID = 2;
}
```

- the protobuf compiler can compile code into c++ .o file and .h headers
- using these structure in C++ is just like other class you defined in C++
- protobuf provide get_ set_ has_ function like has_name()
- protobuf compiler can also compile the code for java, python

```python
student mary;
mary.set_name("mary");
```
Protobuf — a example

caffe reads `solver.prototxt` into a SolverParameter object

### protobuf definition

```protobuf
definition

message SolverParameter {
  optional string train_net = 1; // The proto file for the training net.
  optional string test_net = 2; // The proto file for the testing net.
  // The number of iterations for each test
  optional int32 test_iter = 3 [default = 0];
  // The number of iterations between two test_iter
  optional int32 test_interval = 4 [default = 0];
  // The base learning rate, momentum and the weight decay of the network.
  optional float base_lr = 5; // The base learning rate
  optional float base_flip = 21; // The base flipping rate
  // the number of iterations between displaying info. If display = 0, no info
  // will be displayed.
  optional int32 display = 6;
  optional int32 max_iter = 7; // the maximum number of iterations
  optional string lr_policy = 8; // The learning rate decay policy.
  optional float lr_gamma = 9; // The parameter to compute the learning rate.
  optional float lr_power = 10; // The parameter to compute the learning rate.
}
```

### solver.prototxt

```plaintext
# The train/test net protocol buffer definition
train_net: "examples/mnist/lenet_train.prototxt"

test_net: "examples/mnist/lenet_test.prototxt"

# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 test iterations.
# covering the full 10,000 testing images.

test_iter: 100

# Carry out testing every 500 training iterations.

test_interval: 500

# The base learning rate, momentum and the weight decay of the network.

base_lr: 0.01

momentum: 0.9

weight_decay: 0.0005

# The learning rate policy.

lr_policy: "inv"

gamma: 0.0001

power: 0.75

# Display every 100 iterations.

display: 100

# The maximum number of iterations.

max_iter: 10000

# snapshot intermediate results

snapshot: 5000
```
Adding layers

$CAFFE/src/layers

implement `xx_layer.cpp` and `xx_layer.cu`

- `Forward_cpu`
- `Backward_cpu`
- `Forward_gpu`
- `Backward_gpu`
- `SetUp`
Adding layers

show `inner_product.cpp` and `inner_product.cu`
tuning CNN
a few tips

• Our Goal: fitting the data as much as possible —> making the training cost as small as possible.

• Things that we could tune:
  • learning rate: large learning rate would cause the cost to go bigger and finally go to NaN.
  • Parameter Initialization: Bad initialization would give no gradient over parameters —> no learning occurs.

• How to tune those parameters:
  • monitor the testing cost after each several iterations.
  • monitor the gradient and the value of model parameters (abs mean of each layer).