

ImageNet Recipes from a Practical View

Yangqing Jia
Google Brain

“Where do we start?”

x ilsvrc2012 train			
Name	Size	Type	
▶ n01440764	1,300 items	folder	
▶ n01443537	1,300 items	folder	
▶ n01484850	1,300 items	folder	
▶ n01491361	1,300 items	folder	
▶ n01494475	1,300 items	folder	
▶ n01496331	1,300 items	folder	
▶ n01498041	1,300 items	folder	
▶ n01514668	1,300 items	folder	
▶ n01514859	1,300 items	folder	
▶ n01518878	1,300 items	folder	
▶ n01530575	1,300 items	folder	
▶ n01531178	1,300 items	folder	
▶ n01532829	1,300 items	folder	
▶ n01534433	1,300 items	folder	

x ilsvrc2012 train			
Name	Size	Type	
▼ n01440764	1,300 items	folder	
▶ n01440764_18.JPEG	244.7 kB	JPEG image	
▶ n01440764_36.JPEG	35.1 kB	JPEG image	
▶ n01440764_37.JPEG	73.2 kB	JPEG image	
▶ n01440764_39.JPEG	145.4 kB	JPEG image	
▶ n01440764_44.JPEG	26.7 kB	JPEG image	
▶ n01440764_63.JPEG	15.2 kB	JPEG image	
▶ n01440764_73.JPEG	75.3 kB	JPEG image	
▶ n01440764_78.JPEG	275.2 kB	JPEG image	
▶ n01440764_96.JPEG	100.6 kB	JPEG image	
▶ n01440764_97.JPEG	183.1 kB	JPEG image	
▶ n01440764_105.JPEG	120.3 kB	JPEG image	
▶ n01440764_107.JPEG	2.8 kB	JPEG image	

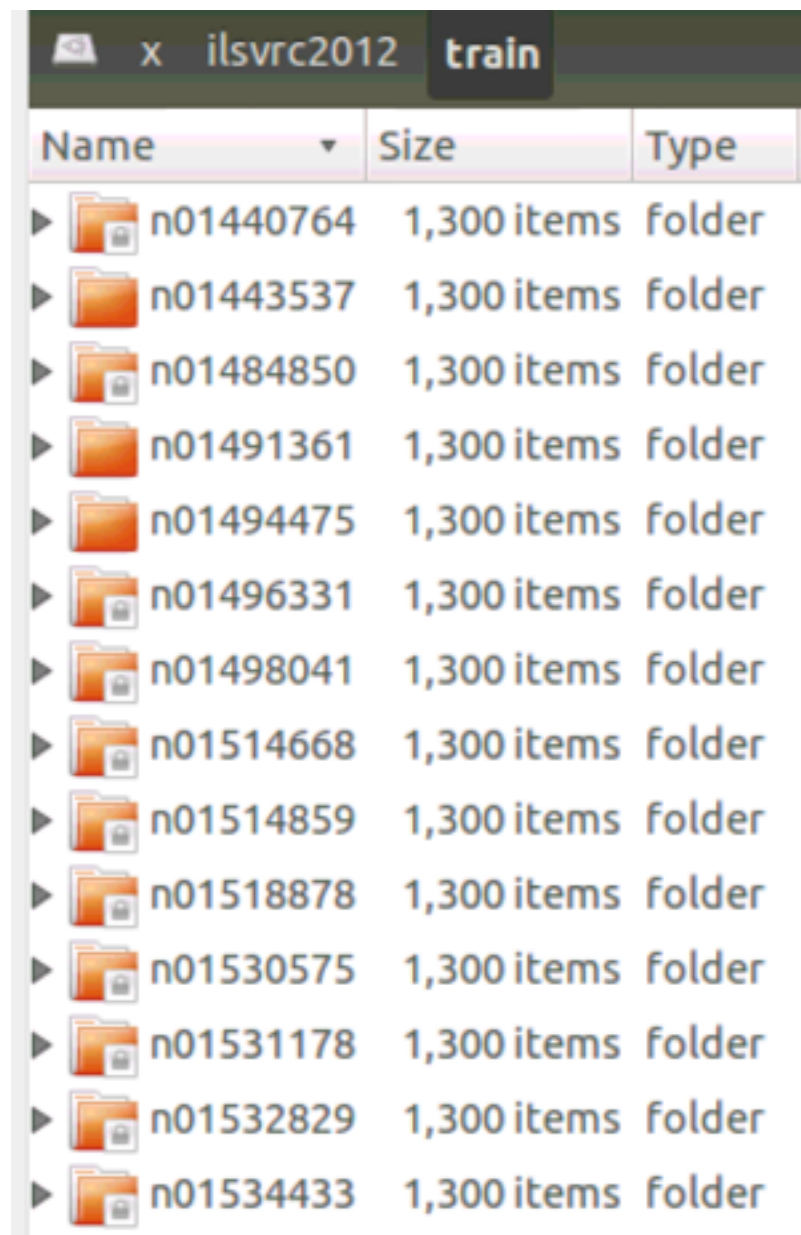
Outline



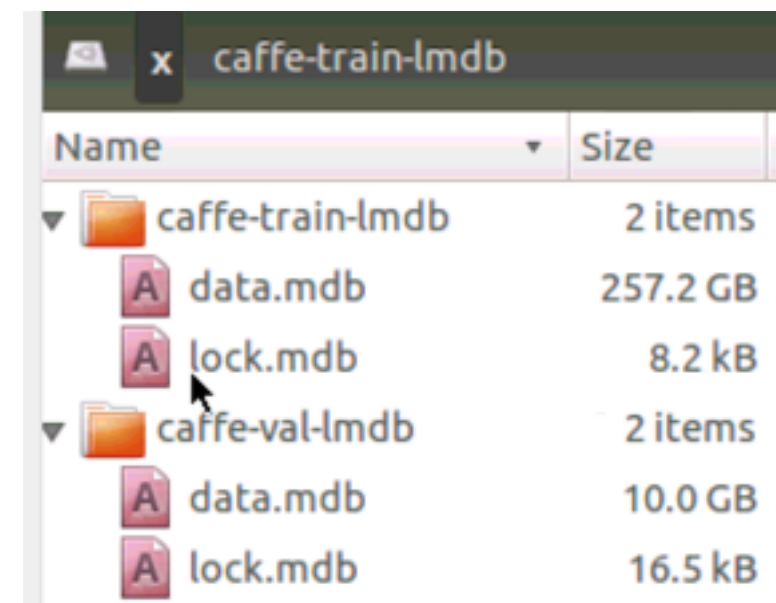
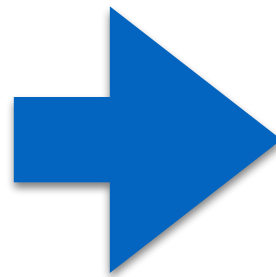
- Data Preparation
- Setting models, training, inspecting results
- Implement My Own Algorithms
- Detection Routines

Preparing Your Data

- Putting them into a sequentially-readable format



Name	Size	Type
n01440764	1,300 items	folder
n01443537	1,300 items	folder
n01484850	1,300 items	folder
n01491361	1,300 items	folder
n01494475	1,300 items	folder
n01496331	1,300 items	folder
n01498041	1,300 items	folder
n01514668	1,300 items	folder
n01514859	1,300 items	folder
n01518878	1,300 items	folder
n01530575	1,300 items	folder
n01531178	1,300 items	folder
n01532829	1,300 items	folder
n01534433	1,300 items	folder



Name	Size
caffe-train-lmdb	2 items
data.mdb	257.2 GB
lock.mdb	8.2 kB
caffe-val-lmdb	2 items
data.mdb	10.0 GB
lock.mdb	16.5 kB

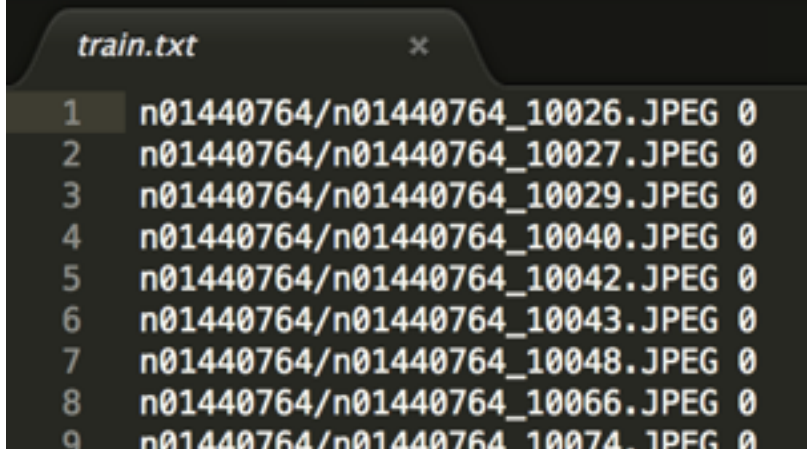
Why do I Need a DB?

- Latency Matters.
- Sequential Read: ~150 MB/s
- Random Read: ~6 MB/s
- AlexNet needs: ~40 MB/s* (200 images/s)
- 8-GPU AlexNet needs: ~320 MB/s

*: Based on 256x256 uncompressed color image, and 5ms/image training speed.
Data is on HDD - SSD is another story

Data Preparation in Caffe

- Create a list of files, together with labels
- Run the preparation script to create a database (leveldb/Imdb)
 - Resizes images
 - Randomly shuffles them
 - Puts them as protocol buffers containing image and label
 - Write them as serialized binary strings

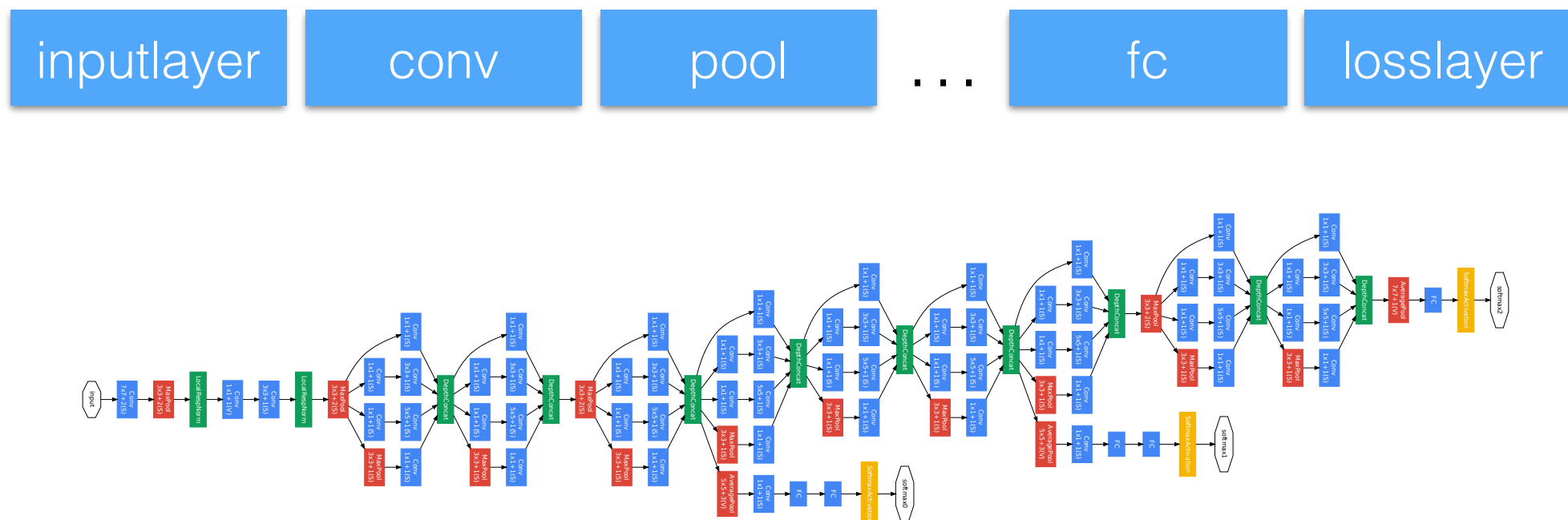


```
train.txt
1 n01440764/n01440764_10026.JPEG 0
2 n01440764/n01440764_10027.JPEG 0
3 n01440764/n01440764_10029.JPEG 0
4 n01440764/n01440764_10040.JPEG 0
5 n01440764/n01440764_10042.JPEG 0
6 n01440764/n01440764_10043.JPEG 0
7 n01440764/n01440764_10048.JPEG 0
8 n01440764/n01440764_10066.JPEG 0
9 n01440764/n01440764_10074.JPEG 0
```

[See examples/imagenet/create_imagenet.sh](#)

Setting up a Model

- A Model is a DAG of layers.



- In Caffe, we use the Google Protocol Buffer format.
*Why? It is cross-platform and cross-language.
But any XML-like language will work.

Write a model in txt

```
name: "CaffeNet"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  ....
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  ...
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
  ...
}
```

- Not the easiest, but doable
- Examples:
<https://github.com/BVLC/caffe/tree/master/models>

Good Thing about ProtoBuf...

Protobuf Definition:

```
message Person {  
  required string name = 1;  
  required int32 id = 2;  
  optional string email = 3;  
}
```

Python example code:

```
p = Person()  
p.name = "John Doe"  
p.id = 1701  
p.email = "doe@g.com"  
s = p.SerializeToString
```

C++ example code:

```
Person p;  
p.set_name("John Doe");  
p.set_id(1701);  
p.set_email("doe@g.com")  
string s = p.SerializeAsString()
```

- An easy way to define structured data across languages
 - Make a model in python, run in C++
 - Inspect trained models more easily

Write a model in Python

```
from caffe import layers as L
from caffe import params as P

def lenet(lmdb, batch_size):
    # our version of LeNet: a series of linear and simple nonlinear transformations
    n = caffe.NetSpec()
    n.data, n.label = L.Data(batch_size=batch_size, backend=P.Data.LMDB, source=lmdb,
                             transform_param=dict(scale=1./255), ntop=2)
    n.conv1 = L.Convolution(n.data, kernel_size=5, num_output=20, weight_filler=dict(type='xavier'))
    n.pool1 = L.Pooling(n.conv1, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.conv2 = L.Convolution(n.pool1, kernel_size=5, num_output=50, weight_filler=dict(type='xavier'))
    n.pool2 = L.Pooling(n.conv2, kernel_size=2, stride=2, pool=P.Pooling.MAX)
    n.ip1 = L.InnerProduct(n.pool2, num_output=500, weight_filler=dict(type='xavier'))
    n.relu1 = L.ReLU(n.ip1, in_place=True)
    n.ip2 = L.InnerProduct(n.relu1, num_output=10, weight_filler=dict(type='xavier'))
    n.loss = L.SoftmaxWithLoss(n.ip2, n.label)
    return n.to_proto()

with open('examples/mnist/lenet_auto_train.prototxt', 'w') as f:
    f.write(str(lenet('examples/mnist/mnist_train_lmdb', 64)))

with open('examples/mnist/lenet_auto_test.prototxt', 'w') as f:
    f.write(str(lenet('examples/mnist/mnist_test_lmdb', 100)))
```

* Caffe Pull Request #2086

Run Training

- Make a Solver Definition (also a protobuffer):

```
net: "models/bvlc_reference_caffenet/train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 100000
display: 20
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix: "models/bvlc_reference_caffenet/caffenet_train"
solver_mode: GPU
```

What the Solver does...

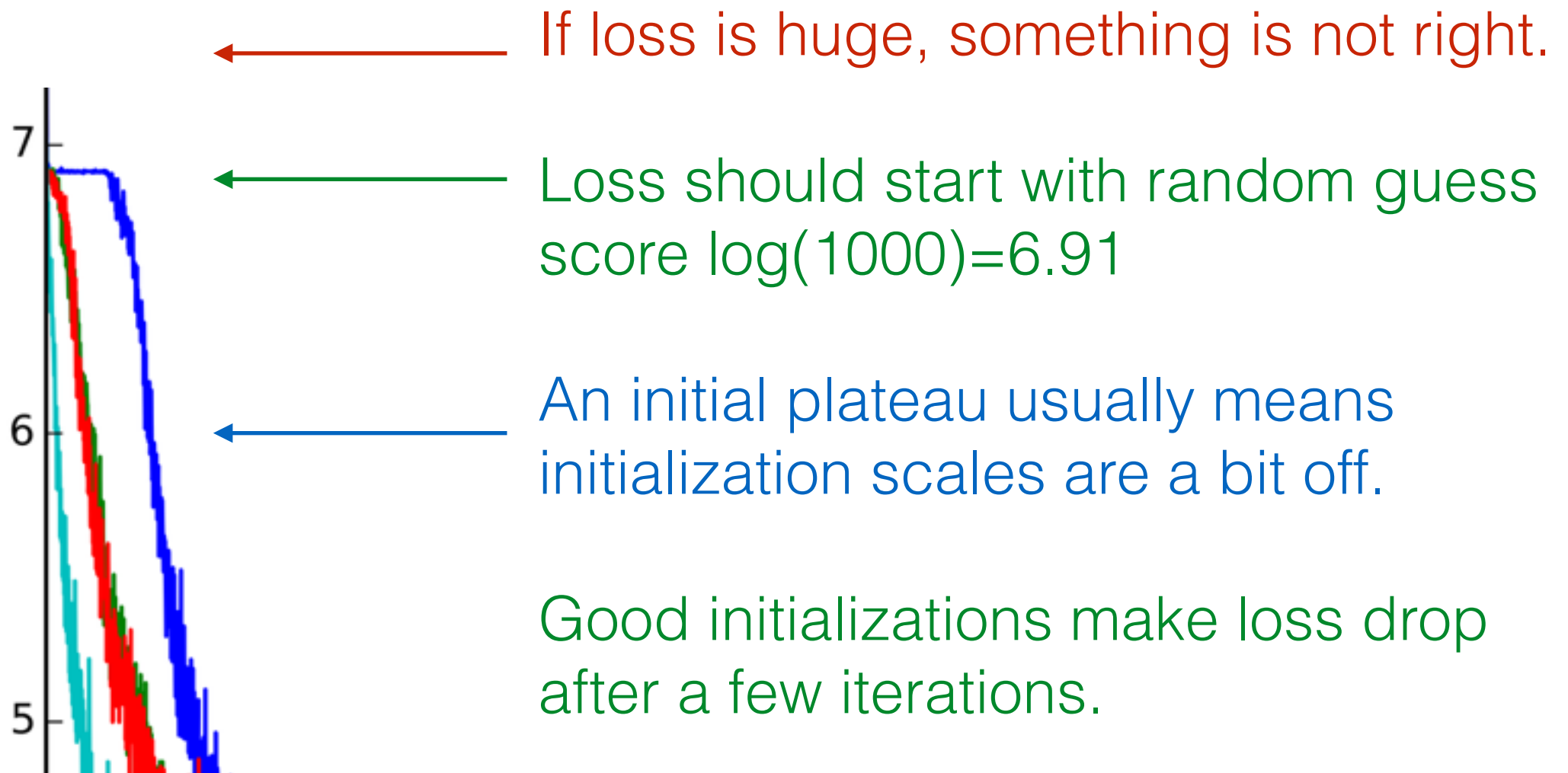
- It instantiates the network either on the CPU or GPU
- It runs a standard SGD (optionally w/ momentum, Adagrad, etc)
- Periodically, it
 - verifies the accuracy on the validation data
 - dumps the current network parameters to disk.

Run Training

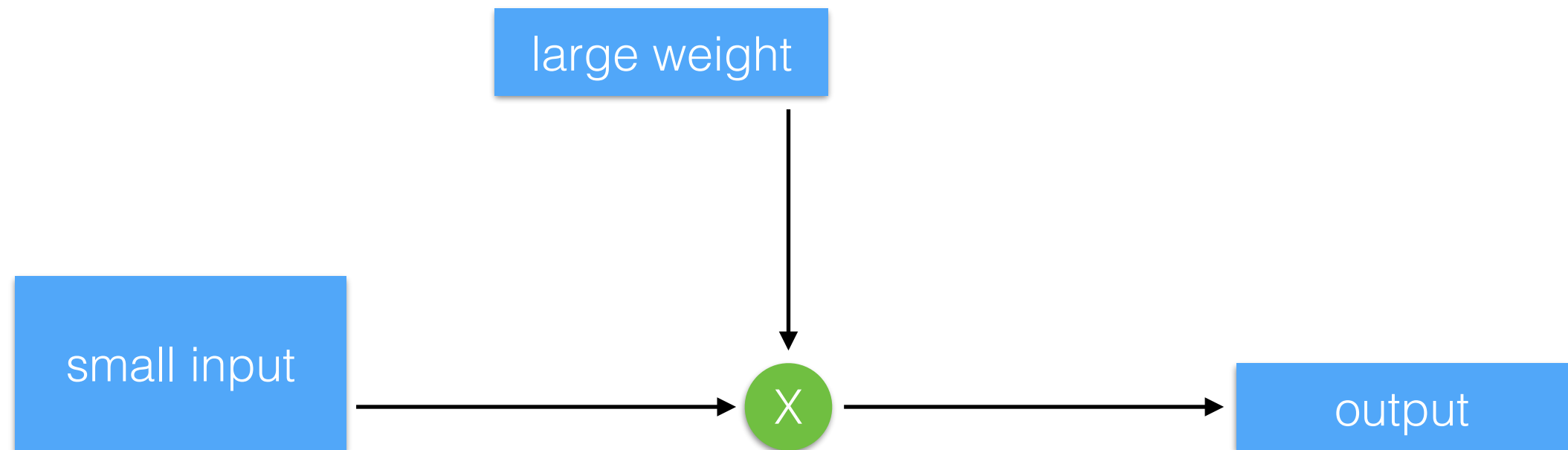
- Train with existing tool:
`FOLDER=models/bvlc_reference_caffenet`
`./build/tools/caffe train \`
`--solver=$FOLDER/solver.prototxt`
- Oops, I tripped over power supply (or my cat did):
`./build/tools/caffe train \`
`--solver=$FOLDER/solver.prototxt \`
`--snapshot=$FOLDER/caffenet_train_10000.solverstate`
- If you would like to write your own solver...
<https://github.com/BVLC/caffe/blob/master/include/caffe/solver.hpp>

Inspecting Progress

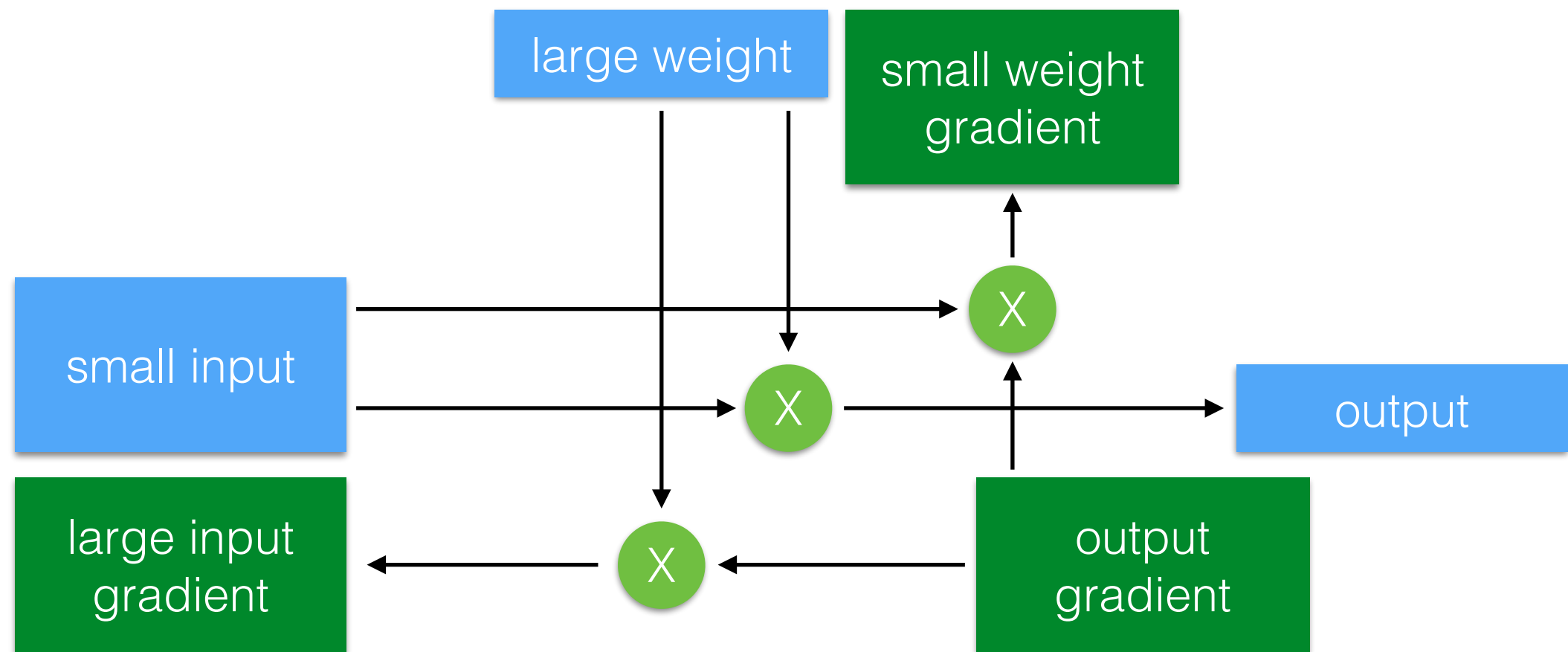
- Make sure you look at the loss, esp. at the beginning.



Why scale Matters?



Why scale Matters?

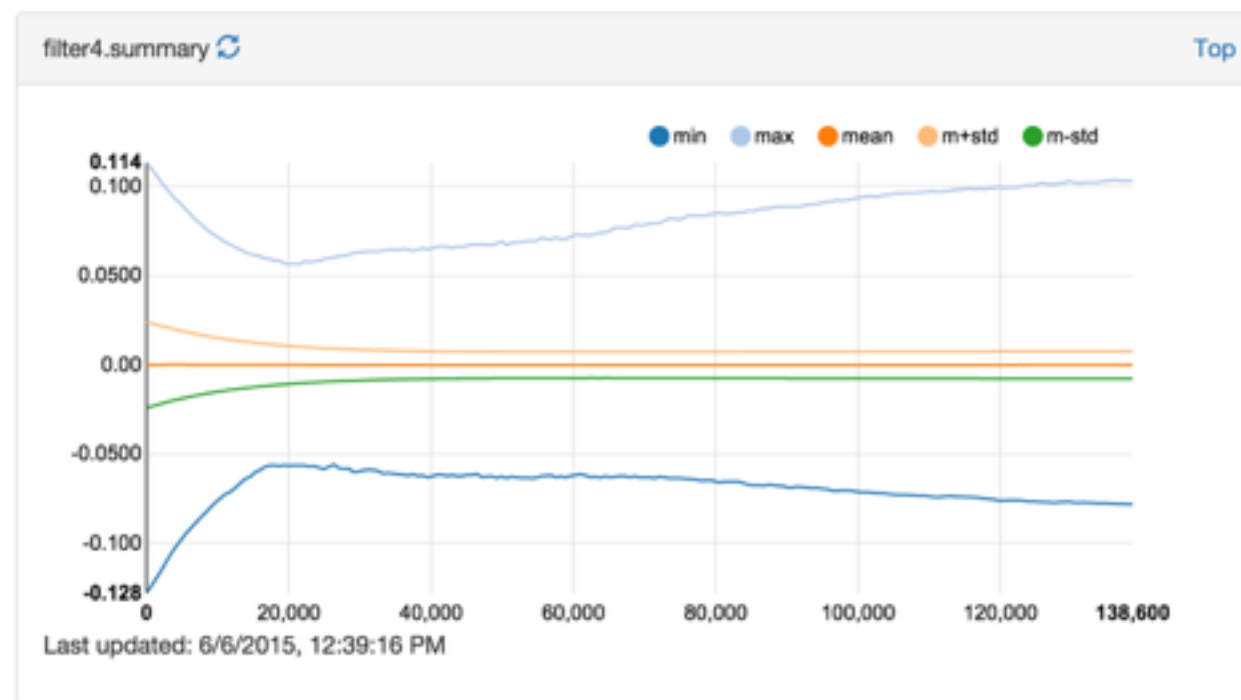


Initialization Gets More Principled

- Batch Normalization [[arxiv](#)]
- Robust Initializations [[arxiv](#)]

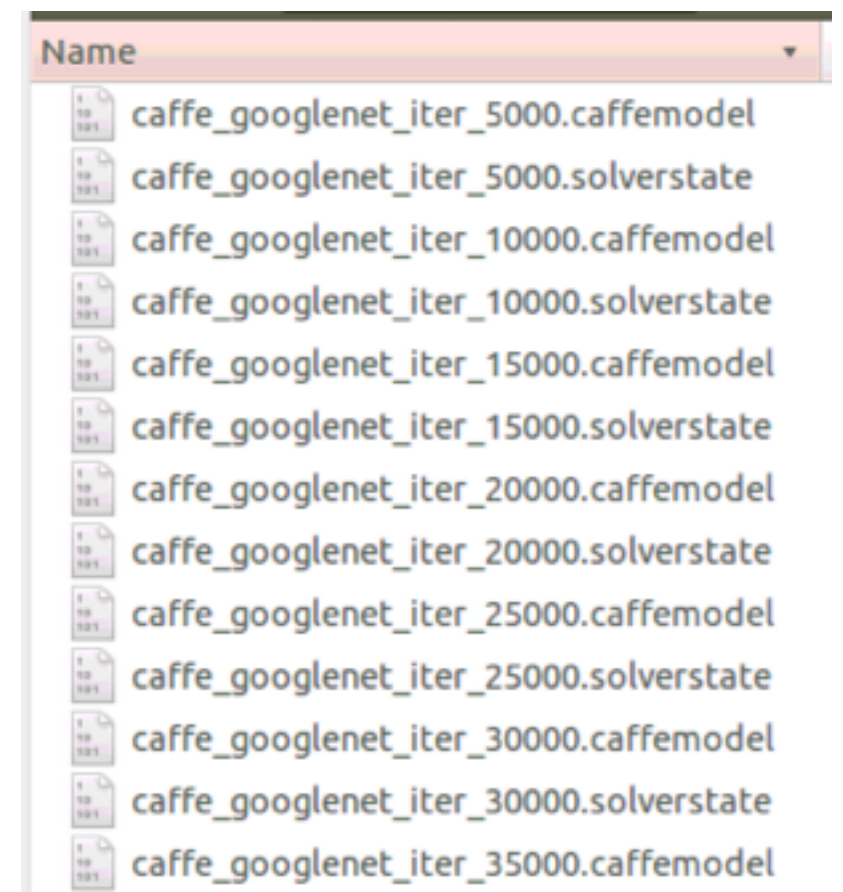
Also Consider...

- Monitor the progress of the parameters as well.



What the Solver Dumps

- *.caffemodel: the trained model
You may want to look at the model to see what it learned.
- *.solverstate: data to restore the solver state (e.g. iteration, momentum, etc.)
These are not the models you are looking for.



Inspecting Model

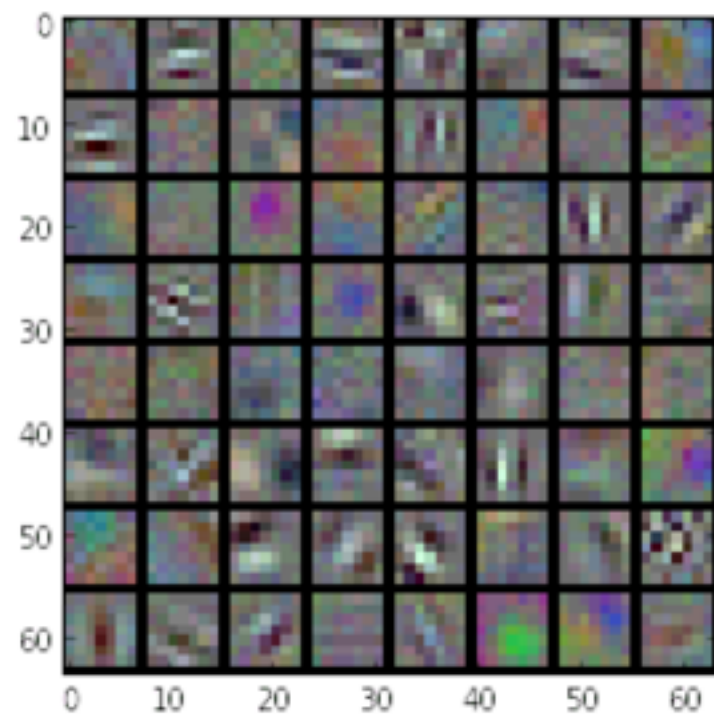
- “What have We Done?” - we dumped a ProtoBuf

```
from caffe.proto import caffe_pb2
model = caffe_pb2.NetParameter()
model.ParseFromString(
    open('blabla.caffemodel').read())
```

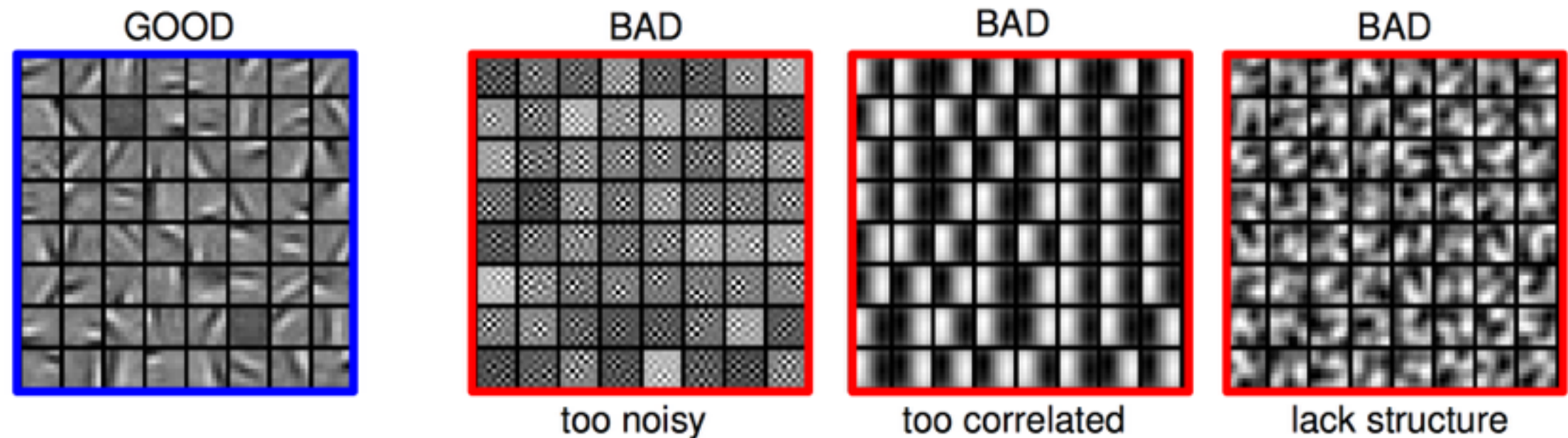
- And you should be able to inspect every piece of the model.

Inspecting Model

```
model = caffe_pb2.NetParameter()
model.ParseFromString(
    open('/x/jiayq/googlenet_for_tutorial/'
        'caffe_googlenet_iter_500000.caffemodel').read())
conv1_layer = model.layers[2]
filters = np.asarray(conv1_layer.blobs[0].data).reshape(
    conv1_layer.blobs[0].num, conv1_layer.blobs[0].channels,
    conv1_layer.blobs[0].height, conv1_layer.blobs[0].width)
vis_filters(filters)
```



Inspecting Model



* Figure borrowed from Ranzato CVPR'14

Inspection with Python

- http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/filter_visualization.ipynb
- Marc'Aurelio Ranzato's [presentation at CVPR'14](#).

Quick Prototyping in Python

- <https://github.com/BVLC/caffe/blob/tutorial/examples/01-learning-lenet.ipynb>
- (This is lenet, but imagenet is similar)

When it doesn't work

- Loss does not decrease.
Learning rate? Initialization?
- Performance is capped
Larger models? Optimization problems?
- Speed too slow
Per-layer Latency? Prefetching? GPU?
- And as always... Bug in the code?
Unit test? Gradient checking?

Implement My Own Layer

- Implement a custom layer
 - Derived class from the Layer base class.
 - `LayerSetUp()`: sets up layer parameters.
 - `Reshape()`: sets output shapes.
 - `{Forward,Backward}_cpu()`: actual computation.
 - `{Forward,Backward}_gpu()`: optional GPU code.
- Example: PReLU
<https://github.com/BVLC/caffe/pull/1940/>

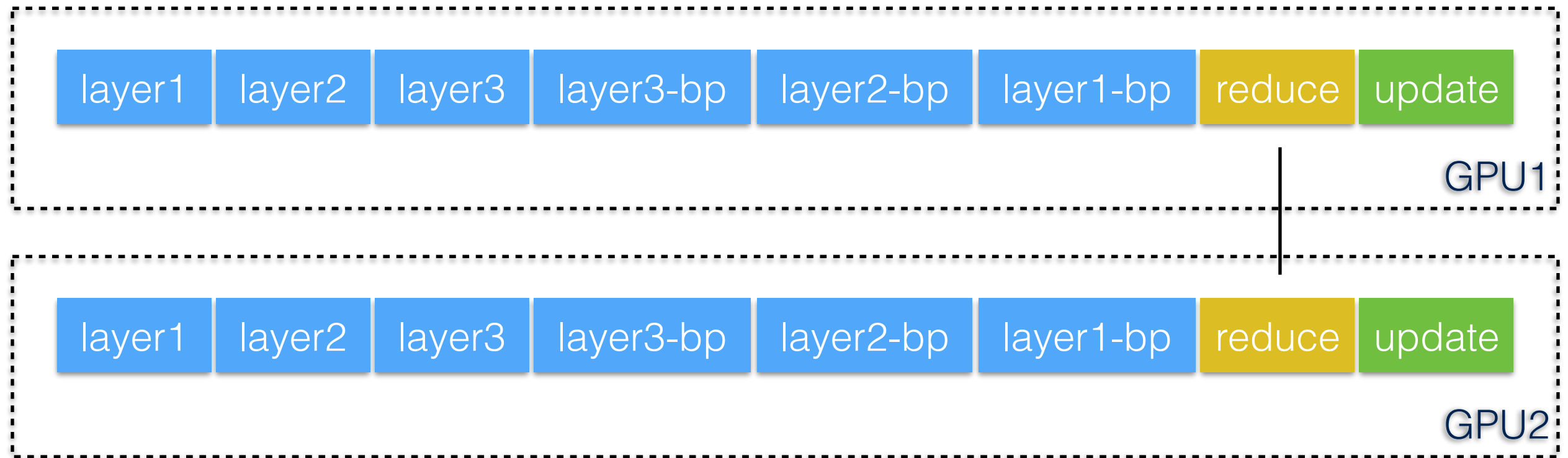
Multi-GPU?

- Essentially, two things: computation and communication.



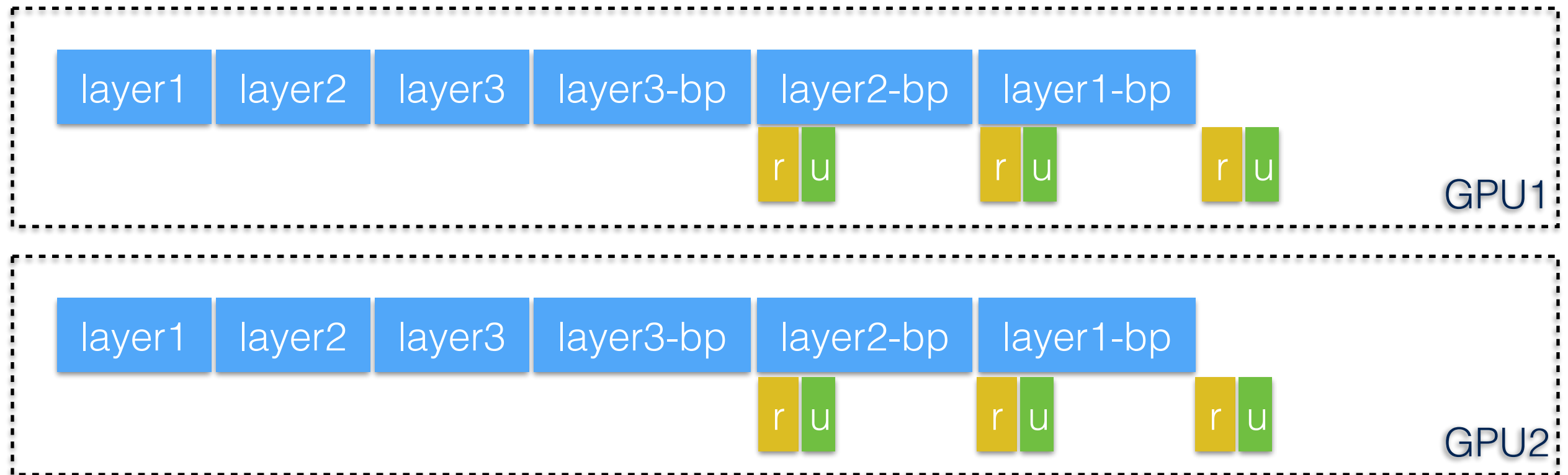
Multi-GPU?

- Essentially, two things: computation and communication.
- Consider MPI + GPUDirect



Latency Matters

- Essentially, two things: computation and communication.
- Pipeline them to hide latency:



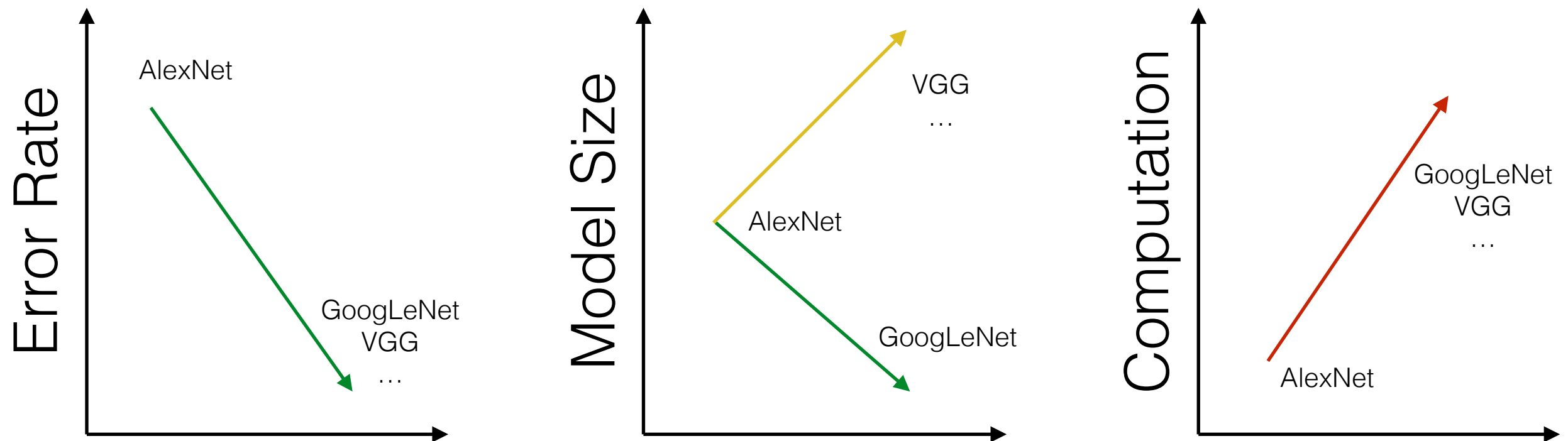
Overall Pipeline



- Some personal experiences...
- Start interactively, use tools like Python
- Be careful about latency
- Inspect model often

Random Thoughts...

- Different criteria for “better” models



Detection

- More or less like classification, but...
 - Need to find bounding boxes.
 - (which means quite some tinkering)
- As of last year, detection is a two-step procedure
 - Bounding box proposal (optionally w/ labels)
 - Prediction on the bounding boxes

A Little More about Detection

- R-CNN: running selective search, and then use Caffe to train classifiers.

Ross's R-CNN code: <https://github.com/rbgirshick/rcnn>

Recent fast R-CNN: <https://github.com/rbgirshick/fast-rcnn>

- MultiBox: running a CNN model to predict the bounding boxes

Essentially the same GoogLeNet, but with a custom loss function that evaluates boxes

- After boxes: collect bounding box images, train classifiers similar to classification.

Random Thoughts

- “One system to rule them all”
Powerful CNN to predict both location and class?
Attention model?

What to Expect...

Lots of hacking. In a good way.



More about Caffe

There is a Caffe tutorial session in the afternoon:

2-6pm, Room 200

<http://tutorial.caffe.berkeleyvision.org/>