caffe (/github/BVLC/caffe/tree/master) / examples (/github/BVLC/caffe/tree/master/examples)

Brewing Logistic Regression then Going Deeper

While Caffe is made for deep networks it can likewise represent "shallow" models like logistic regression for classification. We'll do simple logistic regression on synthetic data that we'll generate and save to HDF5 to feed vectors to Caffe. Once that model is done, we'll add layers to improve accuracy. That's what Caffe is about: define a model, experiment, and then deploy.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import os
os.chdir('..')

import sys
sys.path.insert(0, './python')
import caffe


import os
import h5py
import shutil
import tempfile

import sklearn
import sklearn.datasets
import sklearn.linear_model

import pandas as pd
```

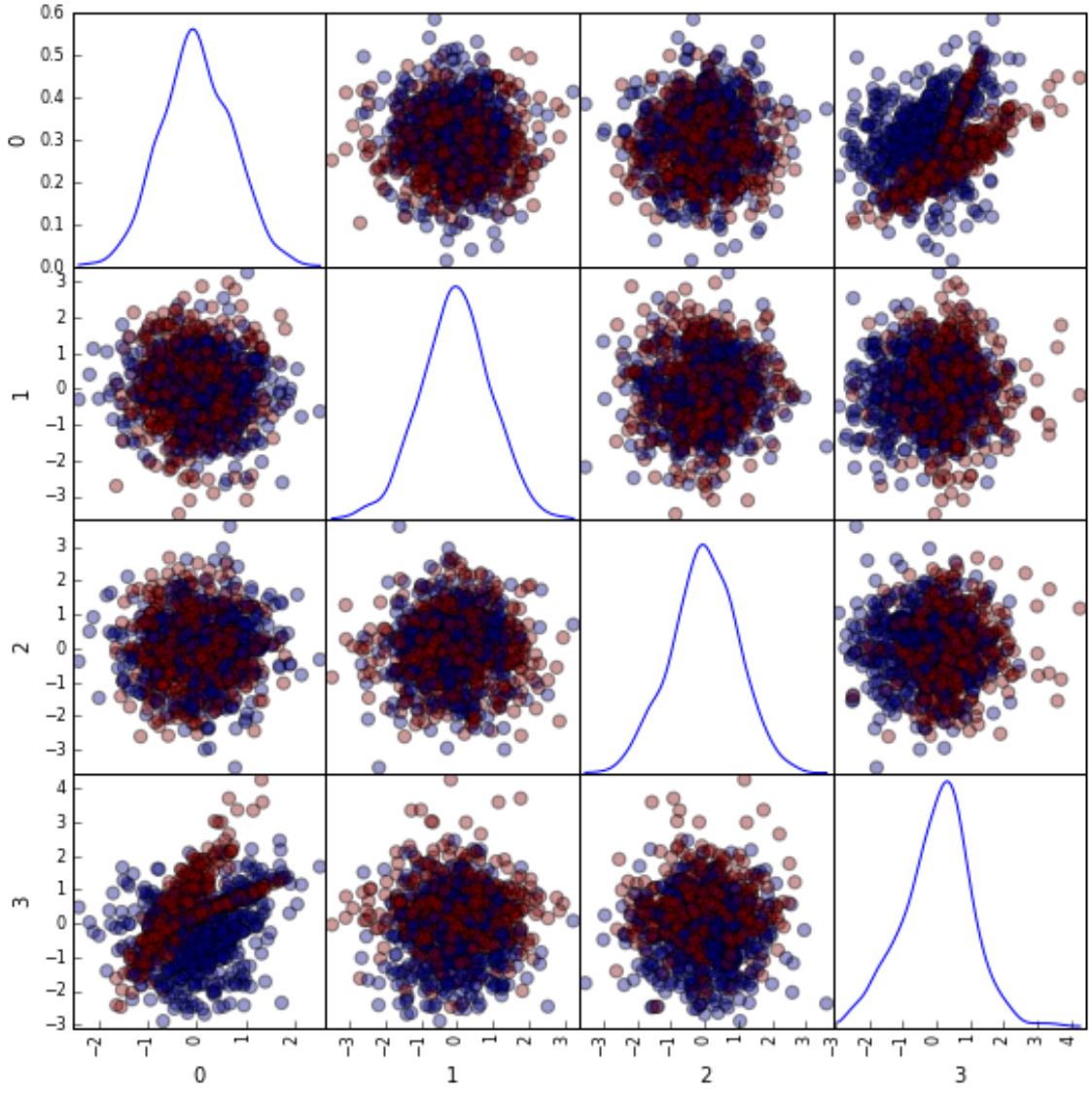
Synthesize a dataset of 10,000 4-vectors for binary classification with 2 informative features and 2 noise features.

In [2]:

```
X, y = sklearn.datasets.make_classification(
    n_samples=10000, n_features=4, n_redundant=0, n_informative=2,
    n_clusters_per_class=2, hypercube=False, random_state=0
)

# Split into train and test
X, Xt, y, yt = sklearn.cross_validation.train_test_split(X, y)

# Visualize sample of the data
ind = np.random.permutation(X.shape[0])[:1000]
df = pd.DataFrame(X[ind])
_ = pd.scatter_matrix(df, figsize=(9, 9), diagonal='kde', marker='o', s=40, alpha=.4, c=
y[ind])
```



Learn and evaluate scikit-learn's logistic regression with stochastic gradient descent (SGD) training. Time and check the classifier's accuracy.

In [3]:

```
%%timeit
# Train and test the scikit-learn SGD logistic regression.
clf = sklearn.linear_model.SGDClassifier(
    loss='log', n_iter=1000, penalty='l2', alpha=5e-4, class_weight='auto')

clf.fit(X, y)
yt_pred = clf.predict(Xt)
print('Accuracy: {:.3f}'.format(sklearn.metrics.accuracy_score(yt, yt_pred)))
```

```
Accuracy: 0.781
Accuracy: 0.781
Accuracy: 0.781
Accuracy: 0.781
1 loop, best of 3: 372 ms per loop
```

Save the dataset to HDF5 for loading in Caffe.

In [4]:

```
# Write out the data to HDF5 files in a temp directory.
# This file is assumed to be caffe_root/examples/hdf5_classification.ipynb
dirname = os.path.abspath('./examples/hdf5_classification/data')
if not os.path.exists(dirname):
    os.makedirs(dirname)

train_filename = os.path.join(dirname, 'train.h5')
test_filename = os.path.join(dirname, 'test.h5')

# HDF5DataLayer source should be a file containing a list of HDF5 filenames.
# To show this off, we'll list the same data file twice.
with h5py.File(train_filename, 'w') as f:
    f['data'] = X
    f['label'] = y.astype(np.float32)
with open(os.path.join(dirname, 'train.txt'), 'w') as f:
    f.write(train_filename + '\n')
    f.write(train_filename + '\n')

# HDF5 is pretty efficient, but can be further compressed.
comp_kwargs = {'compression': 'gzip', 'compression_opts': 1}
with h5py.File(test_filename, 'w') as f:
    f.create_dataset('data', data=Xt, **comp_kwargs)
    f.create_dataset('label', data=yt.astype(np.float32), **comp_kwargs)
with open(os.path.join(dirname, 'test.txt'), 'w') as f:
    f.write(test_filename + '\n')
```

Let's define logistic regression in Caffe through Python net specification. This is a quick and natural way to define nets that sidesteps manually editing the protobuf model.

In [5]:

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

def logreg(hdf5, batch_size):
    # logistic regression: data, matrix multiplication, and 2-class softmax loss
    n = caffe.NetSpec()
    n.data, n.label = L.HDF5Data(batch_size=batch_size, source=hdf5, ntop=2)
    n.ip1 = L.InnerProduct(n.data, num_output=2, weight_filler=dict(type='xavier'))
    n.accuracy = L.Accuracy(n.ip1, n.label)
    n.loss = L.SoftmaxWithLoss(n.ip1, n.label)
    return n.to_proto()

train_net_path = 'examples/hdf5_classification/logreg_auto_train.prototxt'
with open(train_net_path, 'w') as f:
    f.write(str(logreg('examples/hdf5_classification/data/train.txt', 10)))

test_net_path = 'examples/hdf5_classification/logreg_auto_test.prototxt'
with open(test_net_path, 'w') as f:
    f.write(str(logreg('examples/hdf5_classification/data/test.txt', 10)))
```

Now, we'll define our "solver" which trains the network by specifying the locations of the train and test nets we defined above, as well as setting values for various parameters used for learning, display, and "snapshotting".

In [6]:

```
from caffe.proto import caffe_pb2

def solver(train_net_path, test_net_path):
    s = caffe_pb2.SolverParameter()

    # Specify locations of the train and test networks.
    s.train_net = train_net_path
    s.test_net.append(test_net_path)

    s.test_interval = 1000  # Test after every 1000 training iterations.
    s.test_iter.append(250) # Test 250 "batches" each time we test.

    s.max_iter = 10000      # # of times to update the net (training iterations)

    # Set the initial learning rate for stochastic gradient descent (SGD).
    s.base_lr = 0.01

    # Set `lr_policy` to define how the learning rate changes during training.
    # Here, we 'step' the learning rate by multiplying it by a factor `gamma`
    # every `stepsize` iterations.
    s.lr_policy = 'step'
    s.gamma = 0.1
    s.stepsize = 5000

    # Set other optimization parameters. Setting a non-zero `momentum` takes a
    # weighted average of the current gradient and previous gradients to make
    # learning more stable. L2 weight decay regularizes learning, to help prevent
    # the model from overfitting.
    s.momentum = 0.9
    s.weight_decay = 5e-4

    # Display the current training loss and accuracy every 1000 iterations.
    s.display = 1000

    # Snapshots are files used to store networks we've trained.  Here, we'll
    # snapshot every 10K iterations -- just once at the end of training.
    # For larger networks that take longer to train, you may want to set
    # snapshot < max_iter to save the network and training state to disk during
    # optimization, preventing disaster in case of machine crashes, etc.
    s.snapshot = 10000
    s.snapshot_prefix = 'examples/hdf5_classification/data/train'

    # We'll train on the CPU for fair benchmarking against scikit-learn.
    # Changing to GPU should result in much faster training!
    s.solver_mode = caffe_pb2.SolverParameter.CPU

    return s

solver_path = 'examples/hdf5_classification/logreg_solver.prototxt'
with open(solver_path, 'w') as f:
    f.write(str(solver(train_net_path, test_net_path)))
```

Time to learn and evaluate our Caffeinated logistic regression in Python.

In [7]:

```
%%timeit
caffe.set_mode_cpu()
solver = caffe.get_solver(solver_path)
solver.solve()

accuracy = 0
batch_size = solver.test_nets[0].blobs['data'].num
test_iters = int(len(Xt) / batch_size)
for i in range(test_iters):
    solver.test_nets[0].forward()
    accuracy += solver.test_nets[0].blobs['accuracy'].data
accuracy /= test_iters

print("Accuracy: {:.3f}".format(accuracy))
```

```
Accuracy: 0.770
Accuracy: 0.770
Accuracy: 0.770
Accuracy: 0.770
1 loop, best of 3: 195 ms per loop
```

Do the same through the command line interface for detailed output on the model and solving.

In [8]:

```
!./build/tools/caffe train -solver examples/hdf5_classification/logreg_solver.prototxt
```

```
I0224 00:32:03.232779   655 caffe.cpp:178] Use CPU.
I0224 00:32:03.391911   655 solver.cpp:48] Initializing solver from paramet
ers:
train_net: "examples/hdf5_classification/logreg_auto_train.prototxt"
test_net: "examples/hdf5_classification/logreg_auto_test.prototxt"
test_iter: 250
test_interval: 1000
base_lr: 0.01
display: 1000
max_iter: 10000
lr_policy: "step"
gamma: 0.1
momentum: 0.9
weight_decay: 0.0005
stepsize: 5000
snapshot: 10000
snapshot_prefix: "examples/hdf5_classification/data/train"
solver_mode: CPU
I0224 00:32:03.392065   655 solver.cpp:81] Creating training net from train
_net file: examples/hdf5_classification/logreg_auto_train.prototxt
I0224 00:32:03.392215   655 net.cpp:49] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/hdf5_classification/data/train.txt"
    batch_size: 10
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip1"
  bottom: "label"
  top: "accuracy"
}
```

```
  layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "ip1"
    bottom: "label"
    top: "loss"
  }
  I0224 00:32:03.392365   655 layer_factory.hpp:77] Creating layer data
  I0224 00:32:03.392382   655 net.cpp:106] Creating Layer data
  I0224 00:32:03.392395   655 net.cpp:411] data -> data
  I0224 00:32:03.392423   655 net.cpp:411] data -> label
  I0224 00:32:03.392442   655 hdf5_data_layer.cpp:79] Loading list of HDF5 fi
  lenames from: examples/hdf5_classification/data/train.txt
  I0224 00:32:03.392473   655 hdf5_data_layer.cpp:93] Number of HDF5 files: 2
  I0224 00:32:03.393473   655 hdf5.cpp:32] Datatype class: H5T_FLOAT
  I0224 00:32:03.393862   655 net.cpp:150] Setting up data
  I0224 00:32:03.393884   655 net.cpp:157] Top shape: 10 4 (40)
  I0224 00:32:03.393894   655 net.cpp:157] Top shape: 10 (10)
  I0224 00:32:03.393901   655 net.cpp:165] Memory required for data: 200
  I0224 00:32:03.393911   655 layer_factory.hpp:77] Creating layer label_data
  _1_split
  I0224 00:32:03.393924   655 net.cpp:106] Creating Layer label_data_1_split
  I0224 00:32:03.393934   655 net.cpp:454] label_data_1_split <- label
  I0224 00:32:03.393945   655 net.cpp:411] label_data_1_split -> label_data_1
  _split_0
  I0224 00:32:03.393956   655 net.cpp:411] label_data_1_split -> label_data_1
  _split_1
  I0224 00:32:03.393970   655 net.cpp:150] Setting up label_data_1_split
  I0224 00:32:03.393978   655 net.cpp:157] Top shape: 10 (10)
  I0224 00:32:03.393986   655 net.cpp:157] Top shape: 10 (10)
  I0224 00:32:03.393995   655 net.cpp:165] Memory required for data: 280
  I0224 00:32:03.394001   655 layer_factory.hpp:77] Creating layer ip1
  I0224 00:32:03.394012   655 net.cpp:106] Creating Layer ip1
  I0224 00:32:03.394021   655 net.cpp:454] ip1 <- data
  I0224 00:32:03.394029   655 net.cpp:411] ip1 -> ip1
  I0224 00:32:03.394311   655 net.cpp:150] Setting up ip1
  I0224 00:32:03.394323   655 net.cpp:157] Top shape: 10 2 (20)
  I0224 00:32:03.394331   655 net.cpp:165] Memory required for data: 360
  I0224 00:32:03.394348   655 layer_factory.hpp:77] Creating layer ip1_ip1_0_
  split
  I0224 00:32:03.394358   655 net.cpp:106] Creating Layer ip1_ip1_0_split
  I0224 00:32:03.394366   655 net.cpp:454] ip1_ip1_0_split <- ip1
  I0224 00:32:03.394374   655 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split
  _0
  I0224 00:32:03.394386   655 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split
  _1
  I0224 00:32:03.394395   655 net.cpp:150] Setting up ip1_ip1_0_split
  I0224 00:32:03.394404   655 net.cpp:157] Top shape: 10 2 (20)
  I0224 00:32:03.394424   655 net.cpp:157] Top shape: 10 2 (20)
  I0224 00:32:03.394443   655 net.cpp:165] Memory required for data: 520
  I0224 00:32:03.394450   655 layer_factory.hpp:77] Creating layer accuracy
  I0224 00:32:03.394462   655 net.cpp:106] Creating Layer accuracy
  I0224 00:32:03.394479   655 net.cpp:454] accuracy <- ip1_ip1_0_split_0
  I0224 00:32:03.394489   655 net.cpp:454] accuracy <- label_data_1_split_0
  I0224 00:32:03.394497   655 net.cpp:411] accuracy -> accuracy

  I0224 00:32:03.394510   655 net.cpp:150] Setting up accuracy
  I0224 00:32:03.394536   655 net.cpp:157] Top shape: (1)
```

```
I0224 00:32:03.394543   655 net.cpp:165] Memory required for data: 524
I0224 00:32:03.394551   655 layer_factory.hpp:77] Creating layer loss
I0224 00:32:03.394562   655 net.cpp:106] Creating Layer loss
I0224 00:32:03.394569   655 net.cpp:454] loss <- ip1_ip1_0_split_1
I0224 00:32:03.394577   655 net.cpp:454] loss <- label_data_1_split_1
I0224 00:32:03.394587   655 net.cpp:411] loss -> loss
I0224 00:32:03.394603   655 layer_factory.hpp:77] Creating layer loss
I0224 00:32:03.394624   655 net.cpp:150] Setting up loss
I0224 00:32:03.394634   655 net.cpp:157] Top shape: (1)
I0224 00:32:03.394641   655 net.cpp:160]     with loss weight 1
I0224 00:32:03.394659   655 net.cpp:165] Memory required for data: 528
I0224 00:32:03.394665   655 net.cpp:226] loss needs backward computation.
I0224 00:32:03.394673   655 net.cpp:228] accuracy does not need backward co
mputation.
I0224 00:32:03.394682   655 net.cpp:226] ip1_ip1_0_split needs backward com
putation.
I0224 00:32:03.394690   655 net.cpp:226] ip1 needs backward computation.
I0224 00:32:03.394697   655 net.cpp:228] label_data_1_split does not need b
ackward computation.
I0224 00:32:03.394706   655 net.cpp:228] data does not need backward comput
ation.
I0224 00:32:03.394712   655 net.cpp:270] This network produces output accur
acy
I0224 00:32:03.394721   655 net.cpp:270] This network produces output loss
I0224 00:32:03.394731   655 net.cpp:283] Network initialization done.
I0224 00:32:03.394804   655 solver.cpp:181] Creating test net (#0) specifie
d by test_net file: examples/hdf5_classification/logreg_auto_test.prototxt
I0224 00:32:03.394836   655 net.cpp:49] Initializing net from parameters:
state {
  phase: TEST
}
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/hdf5_classification/data/test.txt"
    batch_size: 10
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "accuracy"

  type: "Accuracy"
  bottom: "ip1"
```

```
  bottom: "label"
  top: "accuracy"
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip1"
  bottom: "label"
  top: "loss"
}
I0224 00:32:03.394953   655 layer_factory.hpp:77] Creating layer data
I0224 00:32:03.394964   655 net.cpp:106] Creating Layer data
I0224 00:32:03.394973   655 net.cpp:411] data -> data
I0224 00:32:03.394984   655 net.cpp:411] data -> label
I0224 00:32:03.394994   655 hdf5_data_layer.cpp:79] Loading list of HDF5 fi
lenames from: examples/hdf5_classification/data/test.txt
I0224 00:32:03.395009   655 hdf5_data_layer.cpp:93] Number of HDF5 files: 1
I0224 00:32:03.395937   655 net.cpp:150] Setting up data
I0224 00:32:03.395953   655 net.cpp:157] Top shape: 10 4 (40)
I0224 00:32:03.395963   655 net.cpp:157] Top shape: 10 (10)
I0224 00:32:03.395970   655 net.cpp:165] Memory required for data: 200
I0224 00:32:03.395978   655 layer_factory.hpp:77] Creating layer label_data
_1_split
I0224 00:32:03.395989   655 net.cpp:106] Creating Layer label_data_1_split
I0224 00:32:03.395997   655 net.cpp:454] label_data_1_split <- label
I0224 00:32:03.396005   655 net.cpp:411] label_data_1_split -> label_data_1
_split_0
I0224 00:32:03.396016   655 net.cpp:411] label_data_1_split -> label_data_1
_split_1
I0224 00:32:03.396028   655 net.cpp:150] Setting up label_data_1_split
I0224 00:32:03.396036   655 net.cpp:157] Top shape: 10 (10)
I0224 00:32:03.396044   655 net.cpp:157] Top shape: 10 (10)
I0224 00:32:03.396051   655 net.cpp:165] Memory required for data: 280
I0224 00:32:03.396059   655 layer_factory.hpp:77] Creating layer ip1
I0224 00:32:03.396069   655 net.cpp:106] Creating Layer ip1
I0224 00:32:03.396075   655 net.cpp:454] ip1 <- data
I0224 00:32:03.396085   655 net.cpp:411] ip1 -> ip1
I0224 00:32:03.396100   655 net.cpp:150] Setting up ip1
I0224 00:32:03.396109   655 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:03.396116   655 net.cpp:165] Memory required for data: 360
I0224 00:32:03.396138   655 layer_factory.hpp:77] Creating layer ip1_ip1_0_
split
I0224 00:32:03.396148   655 net.cpp:106] Creating Layer ip1_ip1_0_split
I0224 00:32:03.396157   655 net.cpp:454] ip1_ip1_0_split <- ip1
I0224 00:32:03.396164   655 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split
_0
I0224 00:32:03.396174   655 net.cpp:411] ip1_ip1_0_split -> ip1_ip1_0_split
_1
I0224 00:32:03.396185   655 net.cpp:150] Setting up ip1_ip1_0_split
I0224 00:32:03.396194   655 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:03.396203   655 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:03.396209   655 net.cpp:165] Memory required for data: 520
I0224 00:32:03.396216   655 layer_factory.hpp:77] Creating layer accuracy
I0224 00:32:03.396225   655 net.cpp:106] Creating Layer accuracy
I0224 00:32:03.396234   655 net.cpp:454] accuracy <- ip1_ip1_0_split_0

I0224 00:32:03.396241   655 net.cpp:454] accuracy <- label_data_1_split_0
I0224 00:32:03.396250   655 net.cpp:411] accuracy -> accuracy
```

```
I0224 00:32:03.396260   655 net.cpp:150] Setting up accuracy
I0224 00:32:03.396270   655 net.cpp:157] Top shape: (1)
I0224 00:32:03.396276   655 net.cpp:165] Memory required for data: 524
I0224 00:32:03.396283   655 layer_factory.hpp:77] Creating layer loss
I0224 00:32:03.396291   655 net.cpp:106] Creating Layer loss
I0224 00:32:03.396299   655 net.cpp:454] loss <- ip1_ip1_0_split_1
I0224 00:32:03.396307   655 net.cpp:454] loss <- label_data_1_split_1
I0224 00:32:03.396317   655 net.cpp:411] loss -> loss
I0224 00:32:03.396327   655 layer_factory.hpp:77] Creating layer loss
I0224 00:32:03.396339   655 net.cpp:150] Setting up loss
I0224 00:32:03.396349   655 net.cpp:157] Top shape: (1)
I0224 00:32:03.396356   655 net.cpp:160]     with loss weight 1
I0224 00:32:03.396365   655 net.cpp:165] Memory required for data: 528
I0224 00:32:03.396373   655 net.cpp:226] loss needs backward computation.
I0224 00:32:03.396381   655 net.cpp:228] accuracy does not need backward co
mputation.
I0224 00:32:03.396389   655 net.cpp:226] ip1_ip1_0_split needs backward com
putation.
I0224 00:32:03.396396   655 net.cpp:226] ip1 needs backward computation.
I0224 00:32:03.396404   655 net.cpp:228] label_data_1_split does not need b
ackward computation.
I0224 00:32:03.396412   655 net.cpp:228] data does not need backward comput
ation.
I0224 00:32:03.396420   655 net.cpp:270] This network produces output accur
acy
I0224 00:32:03.396427   655 net.cpp:270] This network produces output loss
I0224 00:32:03.396437   655 net.cpp:283] Network initialization done.
I0224 00:32:03.396455   655 solver.cpp:60] Solver scaffolding done.
I0224 00:32:03.396473   655 caffe.cpp:219] Starting Optimization
I0224 00:32:03.396482   655 solver.cpp:280] Solving
I0224 00:32:03.396489   655 solver.cpp:281] Learning Rate Policy: step
I0224 00:32:03.396499   655 solver.cpp:338] Iteration 0, Testing net (#0)
I0224 00:32:03.932615   655 solver.cpp:406]     Test net output #0: accurac
y = 0.4268
I0224 00:32:03.932656   655 solver.cpp:406]     Test net output #1: loss =
1.33093 (* 1 = 1.33093 loss)
I0224 00:32:03.932723   655 solver.cpp:229] Iteration 0, loss = 1.06081
I0224 00:32:03.932737   655 solver.cpp:245]     Train net output #0: accura
cy = 0.4
I0224 00:32:03.932749   655 solver.cpp:245]     Train net output #1: loss =
 1.06081 (* 1 = 1.06081 loss)
I0224 00:32:03.932765   655 sgd_solver.cpp:106] Iteration 0, lr = 0.01
I0224 00:32:03.945551   655 solver.cpp:338] Iteration 1000, Testing net (#0
)
I0224 00:32:03.948048   655 solver.cpp:406]     Test net output #0: accurac
y = 0.694
I0224 00:32:03.948065   655 solver.cpp:406]     Test net output #1: loss =
0.60406 (* 1 = 0.60406 loss)
I0224 00:32:03.948091   655 solver.cpp:229] Iteration 1000, loss = 0.505853
I0224 00:32:03.948102   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:03.948113   655 solver.cpp:245]     Train net output #1: loss =
 0.505853 (* 1 = 0.505853 loss)
I0224 00:32:03.948122   655 sgd_solver.cpp:106] Iteration 1000, lr = 0.01
I0224 00:32:03.960741   655 solver.cpp:338] Iteration 2000, Testing net (#0

)
I0224 00:32:03.963214   655 solver.cpp:406]     Test net output #0: accurac
```

```
y = 0.7372
I0224 00:32:03.963249   655 solver.cpp:406]     Test net output #1: loss =
0.595267 (* 1 = 0.595267 loss)
I0224 00:32:03.963276   655 solver.cpp:229] Iteration 2000, loss = 0.549211
I0224 00:32:03.963289   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:03.963299   655 solver.cpp:245]     Train net output #1: loss =
 0.549211 (* 1 = 0.549211 loss)
I0224 00:32:03.963309   655 sgd_solver.cpp:106] Iteration 2000, lr = 0.01
I0224 00:32:03.975945   655 solver.cpp:338] Iteration 3000, Testing net (#0
)
I0224 00:32:03.978435   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7732
I0224 00:32:03.978451   655 solver.cpp:406]     Test net output #1: loss =
0.594998 (* 1 = 0.594998 loss)
I0224 00:32:03.978884   655 solver.cpp:229] Iteration 3000, loss = 0.66133
I0224 00:32:03.978911   655 solver.cpp:245]     Train net output #0: accura
cy = 0.8
I0224 00:32:03.978932   655 solver.cpp:245]     Train net output #1: loss =
 0.66133 (* 1 = 0.66133 loss)
I0224 00:32:03.978950   655 sgd_solver.cpp:106] Iteration 3000, lr = 0.01
I0224 00:32:03.992017   655 solver.cpp:338] Iteration 4000, Testing net (#0
)
I0224 00:32:03.994509   655 solver.cpp:406]     Test net output #0: accurac
y = 0.694
I0224 00:32:03.994525   655 solver.cpp:406]     Test net output #1: loss =
0.60406 (* 1 = 0.60406 loss)
I0224 00:32:03.994551   655 solver.cpp:229] Iteration 4000, loss = 0.505853
I0224 00:32:03.994562   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:03.994573   655 solver.cpp:245]     Train net output #1: loss =
 0.505853 (* 1 = 0.505853 loss)
I0224 00:32:03.994583   655 sgd_solver.cpp:106] Iteration 4000, lr = 0.01
I0224 00:32:04.007200   655 solver.cpp:338] Iteration 5000, Testing net (#0
)
I0224 00:32:04.009686   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7372
I0224 00:32:04.009702   655 solver.cpp:406]     Test net output #1: loss =
0.595267 (* 1 = 0.595267 loss)
I0224 00:32:04.009727   655 solver.cpp:229] Iteration 5000, loss = 0.549211
I0224 00:32:04.009738   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:04.009749   655 solver.cpp:245]     Train net output #1: loss =
 0.549211 (* 1 = 0.549211 loss)
I0224 00:32:04.009758   655 sgd_solver.cpp:106] Iteration 5000, lr = 0.001
I0224 00:32:04.022734   655 solver.cpp:338] Iteration 6000, Testing net (#0
)
I0224 00:32:04.025177   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7824
I0224 00:32:04.025193   655 solver.cpp:406]     Test net output #1: loss =
0.593367 (* 1 = 0.593367 loss)
I0224 00:32:04.025545   655 solver.cpp:229] Iteration 6000, loss = 0.654873
I0224 00:32:04.025562   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:04.025573   655 solver.cpp:245]     Train net output #1: loss =

 0.654873 (* 1 = 0.654873 loss)
I0224 00:32:04.025583   655 sgd_solver.cpp:106] Iteration 6000, lr = 0.001
```

```
I0224 00:32:04.038586   655 solver.cpp:338] Iteration 7000, Testing net (#0
)
I0224 00:32:04.041016   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7704
I0224 00:32:04.041033   655 solver.cpp:406]     Test net output #1: loss =
0.593842 (* 1 = 0.593842 loss)
I0224 00:32:04.041059   655 solver.cpp:229] Iteration 7000, loss = 0.46611
I0224 00:32:04.041071   655 solver.cpp:245]     Train net output #0: accura
cy = 0.6
I0224 00:32:04.041082   655 solver.cpp:245]     Train net output #1: loss =
 0.46611 (* 1 = 0.46611 loss)
I0224 00:32:04.041091   655 sgd_solver.cpp:106] Iteration 7000, lr = 0.001
I0224 00:32:04.053722   655 solver.cpp:338] Iteration 8000, Testing net (#0
)
I0224 00:32:04.056171   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7788
I0224 00:32:04.056187   655 solver.cpp:406]     Test net output #1: loss =
0.592847 (* 1 = 0.592847 loss)
I0224 00:32:04.056213   655 solver.cpp:229] Iteration 8000, loss = 0.615126
I0224 00:32:04.056224   655 solver.cpp:245]     Train net output #0: accura
cy = 0.8
I0224 00:32:04.056236   655 solver.cpp:245]     Train net output #1: loss =
 0.615126 (* 1 = 0.615126 loss)
I0224 00:32:04.056244   655 sgd_solver.cpp:106] Iteration 8000, lr = 0.001
I0224 00:32:04.068853   655 solver.cpp:338] Iteration 9000, Testing net (#0
)
I0224 00:32:04.071291   655 solver.cpp:406]     Test net output #0: accurac
y = 0.7808
I0224 00:32:04.071307   655 solver.cpp:406]     Test net output #1: loss =
0.593293 (* 1 = 0.593293 loss)
I0224 00:32:04.071650   655 solver.cpp:229] Iteration 9000, loss = 0.654997
I0224 00:32:04.071666   655 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:04.071677   655 solver.cpp:245]     Train net output #1: loss =
 0.654998 (* 1 = 0.654998 loss)
I0224 00:32:04.071687   655 sgd_solver.cpp:106] Iteration 9000, lr = 0.001
I0224 00:32:04.084717   655 solver.cpp:456] Snapshotting to binary proto fi
le examples/hdf5_classification/data/train_iter_10000.caffemodel
I0224 00:32:04.084885   655 sgd_solver.cpp:273] Snapshotting solver state t
o binary proto file examples/hdf5_classification/data/train_iter_10000.solv
erstate
I0224 00:32:04.084960   655 solver.cpp:318] Iteration 10000, loss = 0.46650
5
I0224 00:32:04.084977   655 solver.cpp:338] Iteration 10000, Testing net (#
0)
I0224 00:32:04.087514   655 solver.cpp:406]     Test net output #0: accurac
y = 0.77
I0224 00:32:04.087532   655 solver.cpp:406]     Test net output #1: loss =
0.593815 (* 1 = 0.593815 loss)
I0224 00:32:04.087541   655 solver.cpp:323] Optimization Done.
I0224 00:32:04.087548   655 caffe.cpp:222] Optimization Done.
```

If you look at output or the `logreg_auto_train.prototxt`, you'll see that the model is simple logistic regression. We can make it a little more advanced by introducing a non-linearity between weights that take the input and weights that give the output -- now we have a two-layer network. That

network is given in `nonlinear_auto_train.prototxt`, and that's the only change made in `nonlinear_logreg_solver.prototxt` which we will now use.

The final accuracy of the new network should be higher than logistic regression!

In [9]:

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

def nonlinear_net(hdf5, batch_size):
    # one small nonlinearity, one leap for model kind
    n = caffe.NetSpec()
    n.data, n.label = L.HDF5Data(batch_size=batch_size, source=hdf5, ntop=2)
    # define a hidden layer of dimension 40
    n.ip1 = L.InnerProduct(n.data, num_output=40, weight_filler=dict(type='xavier'))
    # transform the output through the ReLU (rectified linear) non-linearity
    n.relu1 = L.ReLU(n.ip1, in_place=True)
    # score the (now non-linear) features
    n.ip2 = L.InnerProduct(n.ip1, num_output=2, weight_filler=dict(type='xavier'))
    # same accuracy and loss as before
    n.accuracy = L.Accuracy(n.ip2, n.label)
    n.loss = L.SoftmaxWithLoss(n.ip2, n.label)
    return n.to_proto()

train_net_path = 'examples/hdf5_classification/nonlinear_auto_train.prototxt'
with open(train_net_path, 'w') as f:
    f.write(str(nonlinear_net('examples/hdf5_classification/data/train.txt', 10)))

test_net_path = 'examples/hdf5_classification/nonlinear_auto_test.prototxt'
with open(test_net_path, 'w') as f:
    f.write(str(nonlinear_net('examples/hdf5_classification/data/test.txt', 10)))

solver_path = 'examples/hdf5_classification/nonlinear_logreg_solver.prototxt'
with open(solver_path, 'w') as f:
    f.write(str(solver(train_net_path, test_net_path)))
```

In [10]:

```
%%timeit
caffe.set_mode_cpu()
solver = caffe.get_solver(solver_path)
solver.solve()

accuracy = 0
batch_size = solver.test_nets[0].blobs['data'].num
test_iters = int(len(Xt) / batch_size)
for i in range(test_iters):
    solver.test_nets[0].forward()
    accuracy += solver.test_nets[0].blobs['accuracy'].data
accuracy /= test_iters

print("Accuracy: {:.3f}".format(accuracy))
```

```
Accuracy: 0.838
Accuracy: 0.837
Accuracy: 0.838
Accuracy: 0.834
1 loop, best of 3: 277 ms per loop
```

Do the same through the command line interface for detailed output on the model and solving.

In [11]:

```
!./build/tools/caffe train -solver examples/hdf5_classification/nonlinear_logreg_solver.prototxt
```

```
I0224 00:32:05.654265   658 caffe.cpp:178] Use CPU.
I0224 00:32:05.810444   658 solver.cpp:48] Initializing solver from paramet
ers:
train_net: "examples/hdf5_classification/nonlinear_auto_train.prototxt"
test_net: "examples/hdf5_classification/nonlinear_auto_test.prototxt"
test_iter: 250
test_interval: 1000
base_lr: 0.01
display: 1000
max_iter: 10000
lr_policy: "step"
gamma: 0.1
momentum: 0.9
weight_decay: 0.0005
stepsize: 5000
snapshot: 10000
snapshot_prefix: "examples/hdf5_classification/data/train"
solver_mode: CPU
I0224 00:32:05.810634   658 solver.cpp:81] Creating training net from train
_net file: examples/hdf5_classification/nonlinear_auto_train.prototxt
I0224 00:32:05.810835   658 net.cpp:49] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/hdf5_classification/data/train.txt"
    batch_size: 10
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  inner_product_param {
    num_output: 40
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

```
    layer {
      name: "ip2"
      type: "InnerProduct"
      bottom: "ip1"
      top: "ip2"
      inner_product_param {
        num_output: 2
        weight_filler {
          type: "xavier"
        }
      }
    }
    layer {
      name: "accuracy"
      type: "Accuracy"
      bottom: "ip2"
      bottom: "label"
      top: "accuracy"
    }
    layer {
      name: "loss"
      type: "SoftmaxWithLoss"
      bottom: "ip2"
      bottom: "label"
      top: "loss"
    }
    I0224 00:32:05.811061   658 layer_factory.hpp:77] Creating layer data
    I0224 00:32:05.811079   658 net.cpp:106] Creating Layer data
    I0224 00:32:05.811092   658 net.cpp:411] data -> data
    I0224 00:32:05.811121   658 net.cpp:411] data -> label
    I0224 00:32:05.811143   658 hdf5_data_layer.cpp:79] Loading list of HDF5 fi
    lenames from: examples/hdf5_classification/data/train.txt
    I0224 00:32:05.811189   658 hdf5_data_layer.cpp:93] Number of HDF5 files: 2
    I0224 00:32:05.812254   658 hdf5.cpp:32] Datatype class: H5T_FLOAT
    I0224 00:32:05.812677   658 net.cpp:150] Setting up data
    I0224 00:32:05.812705   658 net.cpp:157] Top shape: 10 4 (40)
    I0224 00:32:05.812721   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.812729   658 net.cpp:165] Memory required for data: 200
    I0224 00:32:05.812739   658 layer_factory.hpp:77] Creating layer label_data
    _1_split
    I0224 00:32:05.812752   658 net.cpp:106] Creating Layer label_data_1_split
    I0224 00:32:05.812762   658 net.cpp:454] label_data_1_split <- label
    I0224 00:32:05.812774   658 net.cpp:411] label_data_1_split -> label_data_1
    _split_0
    I0224 00:32:05.812785   658 net.cpp:411] label_data_1_split -> label_data_1
    _split_1
    I0224 00:32:05.812798   658 net.cpp:150] Setting up label_data_1_split
    I0224 00:32:05.812808   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.812816   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.812824   658 net.cpp:165] Memory required for data: 280
    I0224 00:32:05.812831   658 layer_factory.hpp:77] Creating layer ip1
    I0224 00:32:05.812841   658 net.cpp:106] Creating Layer ip1
    I0224 00:32:05.812849   658 net.cpp:454] ip1 <- data
    I0224 00:32:05.812860   658 net.cpp:411] ip1 -> ip1
    I0224 00:32:05.813179   658 net.cpp:150] Setting up ip1
    I0224 00:32:05.813196   658 net.cpp:157] Top shape: 10 40 (400)

    I0224 00:32:05.813210   658 net.cpp:165] Memory required for data: 1880
```

```
I0224 00:32:05.813230   658 layer_factory.hpp:77] Creating layer relu1
I0224 00:32:05.813241   658 net.cpp:106] Creating Layer relu1
I0224 00:32:05.813251   658 net.cpp:454] relu1 <- ip1
I0224 00:32:05.813258   658 net.cpp:397] relu1 -> ip1 (in-place)
I0224 00:32:05.813271   658 net.cpp:150] Setting up relu1
I0224 00:32:05.813279   658 net.cpp:157] Top shape: 10 40 (400)
I0224 00:32:05.813287   658 net.cpp:165] Memory required for data: 3480
I0224 00:32:05.813294   658 layer_factory.hpp:77] Creating layer ip2
I0224 00:32:05.813304   658 net.cpp:106] Creating Layer ip2
I0224 00:32:05.813313   658 net.cpp:454] ip2 <- ip1
I0224 00:32:05.813321   658 net.cpp:411] ip2 -> ip2
I0224 00:32:05.813336   658 net.cpp:150] Setting up ip2
I0224 00:32:05.813345   658 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:05.813379   658 net.cpp:165] Memory required for data: 3560
I0224 00:32:05.813401   658 layer_factory.hpp:77] Creating layer ip2_ip2_0_
split
I0224 00:32:05.813417   658 net.cpp:106] Creating Layer ip2_ip2_0_split
I0224 00:32:05.813426   658 net.cpp:454] ip2_ip2_0_split <- ip2
I0224 00:32:05.813434   658 net.cpp:411] ip2_ip2_0_split -> ip2_ip2_0_split
_0
I0224 00:32:05.813446   658 net.cpp:411] ip2_ip2_0_split -> ip2_ip2_0_split
_1
I0224 00:32:05.813457   658 net.cpp:150] Setting up ip2_ip2_0_split
I0224 00:32:05.813465   658 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:05.813473   658 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:05.813480   658 net.cpp:165] Memory required for data: 3720
I0224 00:32:05.813488   658 layer_factory.hpp:77] Creating layer accuracy
I0224 00:32:05.813499   658 net.cpp:106] Creating Layer accuracy
I0224 00:32:05.813508   658 net.cpp:454] accuracy <- ip2_ip2_0_split_0
I0224 00:32:05.813515   658 net.cpp:454] accuracy <- label_data_1_split_0
I0224 00:32:05.813524   658 net.cpp:411] accuracy -> accuracy
I0224 00:32:05.813539   658 net.cpp:150] Setting up accuracy
I0224 00:32:05.813547   658 net.cpp:157] Top shape: (1)
I0224 00:32:05.813555   658 net.cpp:165] Memory required for data: 3724
I0224 00:32:05.813565   658 layer_factory.hpp:77] Creating layer loss
I0224 00:32:05.813585   658 net.cpp:106] Creating Layer loss
I0224 00:32:05.813599   658 net.cpp:454] loss <- ip2_ip2_0_split_1
I0224 00:32:05.813616   658 net.cpp:454] loss <- label_data_1_split_1
I0224 00:32:05.813627   658 net.cpp:411] loss -> loss
I0224 00:32:05.813642   658 layer_factory.hpp:77] Creating layer loss
I0224 00:32:05.813663   658 net.cpp:150] Setting up loss
I0224 00:32:05.813671   658 net.cpp:157] Top shape: (1)
I0224 00:32:05.813679   658 net.cpp:160]     with loss weight 1
I0224 00:32:05.813695   658 net.cpp:165] Memory required for data: 3728
I0224 00:32:05.813704   658 net.cpp:226] loss needs backward computation.
I0224 00:32:05.813712   658 net.cpp:228] accuracy does not need backward co
mputation.
I0224 00:32:05.813720   658 net.cpp:226] ip2_ip2_0_split needs backward com
putation.
I0224 00:32:05.813729   658 net.cpp:226] ip2 needs backward computation.
I0224 00:32:05.813735   658 net.cpp:226] relu1 needs backward computation.
I0224 00:32:05.813743   658 net.cpp:226] ip1 needs backward computation.
I0224 00:32:05.813751   658 net.cpp:228] label_data_1_split does not need b
ackward computation.
I0224 00:32:05.813760   658 net.cpp:228] data does not need backward comput
ation.

I0224 00:32:05.813772   658 net.cpp:270] This network produces output accur
```

```
acy
I0224 00:32:05.813787   658 net.cpp:270] This network produces output loss
I0224 00:32:05.813809   658 net.cpp:283] Network initialization done.
I0224 00:32:05.813905   658 solver.cpp:181] Creating test net (#0) specifie
d by test_net file: examples/hdf5_classification/nonlinear_auto_test.protot
xt
I0224 00:32:05.813944   658 net.cpp:49] Initializing net from parameters:
state {
  phase: TEST
}
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/hdf5_classification/data/test.txt"
    batch_size: 10
  }
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  inner_product_param {
    num_output: 40
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  inner_product_param {
    num_output: 2
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"

}
```

```
    layer {
      name: "loss"
      type: "SoftmaxWithLoss"
      bottom: "ip2"
      bottom: "label"
      top: "loss"
    }
    I0224 00:32:05.814131   658 layer_factory.hpp:77] Creating layer data
    I0224 00:32:05.814142   658 net.cpp:106] Creating Layer data
    I0224 00:32:05.814152   658 net.cpp:411] data -> data
    I0224 00:32:05.814162   658 net.cpp:411] data -> label
    I0224 00:32:05.814180   658 hdf5_data_layer.cpp:79] Loading list of HDF5 fi
    lenames from: examples/hdf5_classification/data/test.txt
    I0224 00:32:05.814220   658 hdf5_data_layer.cpp:93] Number of HDF5 files: 1
    I0224 00:32:05.815207   658 net.cpp:150] Setting up data
    I0224 00:32:05.815227   658 net.cpp:157] Top shape: 10 4 (40)
    I0224 00:32:05.815243   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.815253   658 net.cpp:165] Memory required for data: 200
    I0224 00:32:05.815260   658 layer_factory.hpp:77] Creating layer label_data
    _1_split
    I0224 00:32:05.815270   658 net.cpp:106] Creating Layer label_data_1_split
    I0224 00:32:05.815279   658 net.cpp:454] label_data_1_split <- label
    I0224 00:32:05.815287   658 net.cpp:411] label_data_1_split -> label_data_1
    _split_0
    I0224 00:32:05.815299   658 net.cpp:411] label_data_1_split -> label_data_1
    _split_1
    I0224 00:32:05.815310   658 net.cpp:150] Setting up label_data_1_split
    I0224 00:32:05.815318   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.815326   658 net.cpp:157] Top shape: 10 (10)
    I0224 00:32:05.815335   658 net.cpp:165] Memory required for data: 280
    I0224 00:32:05.815341   658 layer_factory.hpp:77] Creating layer ip1
    I0224 00:32:05.815351   658 net.cpp:106] Creating Layer ip1
    I0224 00:32:05.815358   658 net.cpp:454] ip1 <- data
    I0224 00:32:05.815367   658 net.cpp:411] ip1 -> ip1
    I0224 00:32:05.815383   658 net.cpp:150] Setting up ip1
    I0224 00:32:05.815398   658 net.cpp:157] Top shape: 10 40 (400)
    I0224 00:32:05.815413   658 net.cpp:165] Memory required for data: 1880
    I0224 00:32:05.815435   658 layer_factory.hpp:77] Creating layer relu1
    I0224 00:32:05.815450   658 net.cpp:106] Creating Layer relu1
    I0224 00:32:05.815459   658 net.cpp:454] relu1 <- ip1
    I0224 00:32:05.815469   658 net.cpp:397] relu1 -> ip1 (in-place)
    I0224 00:32:05.815479   658 net.cpp:150] Setting up relu1
    I0224 00:32:05.815486   658 net.cpp:157] Top shape: 10 40 (400)
    I0224 00:32:05.815495   658 net.cpp:165] Memory required for data: 3480
    I0224 00:32:05.815501   658 layer_factory.hpp:77] Creating layer ip2
    I0224 00:32:05.815510   658 net.cpp:106] Creating Layer ip2
    I0224 00:32:05.815518   658 net.cpp:454] ip2 <- ip1
    I0224 00:32:05.815527   658 net.cpp:411] ip2 -> ip2
    I0224 00:32:05.815542   658 net.cpp:150] Setting up ip2
    I0224 00:32:05.815551   658 net.cpp:157] Top shape: 10 2 (20)
    I0224 00:32:05.815559   658 net.cpp:165] Memory required for data: 3560
    I0224 00:32:05.815570   658 layer_factory.hpp:77] Creating layer ip2_ip2_0_
    split
    I0224 00:32:05.815579   658 net.cpp:106] Creating Layer ip2_ip2_0_split
    I0224 00:32:05.815587   658 net.cpp:454] ip2_ip2_0_split <- ip2
    I0224 00:32:05.815600   658 net.cpp:411] ip2_ip2_0_split -> ip2_ip2_0_split

    _0
```

```
I0224 00:32:05.815619   658 net.cpp:411] ip2_ip2_0_split -> ip2_ip2_0_split
_1
I0224 00:32:05.815640   658 net.cpp:150] Setting up ip2_ip2_0_split
I0224 00:32:05.815654   658 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:05.815662   658 net.cpp:157] Top shape: 10 2 (20)
I0224 00:32:05.815670   658 net.cpp:165] Memory required for data: 3720
I0224 00:32:05.815677   658 layer_factory.hpp:77] Creating layer accuracy
I0224 00:32:05.815685   658 net.cpp:106] Creating Layer accuracy
I0224 00:32:05.815693   658 net.cpp:454] accuracy <- ip2_ip2_0_split_0
I0224 00:32:05.815702   658 net.cpp:454] accuracy <- label_data_1_split_0
I0224 00:32:05.815711   658 net.cpp:411] accuracy -> accuracy
I0224 00:32:05.815722   658 net.cpp:150] Setting up accuracy
I0224 00:32:05.815732   658 net.cpp:157] Top shape: (1)
I0224 00:32:05.815738   658 net.cpp:165] Memory required for data: 3724
I0224 00:32:05.815747   658 layer_factory.hpp:77] Creating layer loss
I0224 00:32:05.815754   658 net.cpp:106] Creating Layer loss
I0224 00:32:05.815762   658 net.cpp:454] loss <- ip2_ip2_0_split_1
I0224 00:32:05.815770   658 net.cpp:454] loss <- label_data_1_split_1
I0224 00:32:05.815779   658 net.cpp:411] loss -> loss
I0224 00:32:05.815790   658 layer_factory.hpp:77] Creating layer loss
I0224 00:32:05.815811   658 net.cpp:150] Setting up loss
I0224 00:32:05.815829   658 net.cpp:157] Top shape: (1)
I0224 00:32:05.815843   658 net.cpp:160]     with loss weight 1
I0224 00:32:05.815867   658 net.cpp:165] Memory required for data: 3728
I0224 00:32:05.815876   658 net.cpp:226] loss needs backward computation.
I0224 00:32:05.815884   658 net.cpp:228] accuracy does not need backward co
mputation.
I0224 00:32:05.815892   658 net.cpp:226] ip2_ip2_0_split needs backward com
putation.
I0224 00:32:05.815901   658 net.cpp:226] ip2 needs backward computation.
I0224 00:32:05.815908   658 net.cpp:226] relu1 needs backward computation.
I0224 00:32:05.815915   658 net.cpp:226] ip1 needs backward computation.
I0224 00:32:05.815923   658 net.cpp:228] label_data_1_split does not need b
ackward computation.
I0224 00:32:05.815932   658 net.cpp:228] data does not need backward comput
ation.
I0224 00:32:05.815938   658 net.cpp:270] This network produces output accur
acy
I0224 00:32:05.815946   658 net.cpp:270] This network produces output loss
I0224 00:32:05.815958   658 net.cpp:283] Network initialization done.
I0224 00:32:05.815978   658 solver.cpp:60] Solver scaffolding done.
I0224 00:32:05.816000   658 caffe.cpp:219] Starting Optimization
I0224 00:32:05.816016   658 solver.cpp:280] Solving
I0224 00:32:05.816030   658 solver.cpp:281] Learning Rate Policy: step
I0224 00:32:05.816048   658 solver.cpp:338] Iteration 0, Testing net (#0)
I0224 00:32:05.831967   658 solver.cpp:406]     Test net output #0: accurac
y = 0.4464
I0224 00:32:05.832033   658 solver.cpp:406]     Test net output #1: loss =
0.909841 (* 1 = 0.909841 loss)
I0224 00:32:05.832186   658 solver.cpp:229] Iteration 0, loss = 0.798509
I0224 00:32:05.832218   658 solver.cpp:245]     Train net output #0: accura
cy = 0.6
I0224 00:32:05.832247   658 solver.cpp:245]     Train net output #1: loss =
 0.798509 (* 1 = 0.798509 loss)
I0224 00:32:05.832281   658 sgd_solver.cpp:106] Iteration 0, lr = 0.01
I0224 00:32:05.859506   658 solver.cpp:338] Iteration 1000, Testing net (#0

)
```

```
I0224 00:32:05.862799   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8156
I0224 00:32:05.862818   658 solver.cpp:406]     Test net output #1: loss =
0.44259 (* 1 = 0.44259 loss)
I0224 00:32:05.862853   658 solver.cpp:229] Iteration 1000, loss = 0.537015
I0224 00:32:05.862864   658 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:05.862875   658 solver.cpp:245]     Train net output #1: loss =
 0.537015 (* 1 = 0.537015 loss)
I0224 00:32:05.862885   658 sgd_solver.cpp:106] Iteration 1000, lr = 0.01
I0224 00:32:05.883155   658 solver.cpp:338] Iteration 2000, Testing net (#0
)
I0224 00:32:05.886435   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8116
I0224 00:32:05.886451   658 solver.cpp:406]     Test net output #1: loss =
0.434079 (* 1 = 0.434079 loss)
I0224 00:32:05.886484   658 solver.cpp:229] Iteration 2000, loss = 0.43109
I0224 00:32:05.886497   658 solver.cpp:245]     Train net output #0: accura
cy = 0.9
I0224 00:32:05.886508   658 solver.cpp:245]     Train net output #1: loss =
 0.43109 (* 1 = 0.43109 loss)
I0224 00:32:05.886518   658 sgd_solver.cpp:106] Iteration 2000, lr = 0.01
I0224 00:32:05.907243   658 solver.cpp:338] Iteration 3000, Testing net (#0
)
I0224 00:32:05.910521   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8168
I0224 00:32:05.910537   658 solver.cpp:406]     Test net output #1: loss =
0.425661 (* 1 = 0.425661 loss)
I0224 00:32:05.910905   658 solver.cpp:229] Iteration 3000, loss = 0.430245
I0224 00:32:05.910922   658 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:05.910933   658 solver.cpp:245]     Train net output #1: loss =
 0.430245 (* 1 = 0.430245 loss)
I0224 00:32:05.910943   658 sgd_solver.cpp:106] Iteration 3000, lr = 0.01
I0224 00:32:05.931205   658 solver.cpp:338] Iteration 4000, Testing net (#0
)
I0224 00:32:05.934479   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8324
I0224 00:32:05.934496   658 solver.cpp:406]     Test net output #1: loss =
0.404891 (* 1 = 0.404891 loss)
I0224 00:32:05.934530   658 solver.cpp:229] Iteration 4000, loss = 0.628955
I0224 00:32:05.934542   658 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:05.934553   658 solver.cpp:245]     Train net output #1: loss =
 0.628955 (* 1 = 0.628955 loss)
I0224 00:32:05.934583   658 sgd_solver.cpp:106] Iteration 4000, lr = 0.01
I0224 00:32:05.955108   658 solver.cpp:338] Iteration 5000, Testing net (#0
)
I0224 00:32:05.958377   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8364
I0224 00:32:05.958395   658 solver.cpp:406]     Test net output #1: loss =
0.404235 (* 1 = 0.404235 loss)
I0224 00:32:05.958432   658 solver.cpp:229] Iteration 5000, loss = 0.394939
I0224 00:32:05.958444   658 solver.cpp:245]     Train net output #0: accura
cy = 0.9
I0224 00:32:05.958456   658 solver.cpp:245]     Train net output #1: loss =

 0.39494 (* 1 = 0.39494 loss)
```

```
I0224 00:32:05.958466   658 sgd_solver.cpp:106] Iteration 5000, lr = 0.001
I0224 00:32:05.978703   658 solver.cpp:338] Iteration 6000, Testing net (#0
)
I0224 00:32:05.981973   658 solver.cpp:406]     Test net output #0: accurac
y = 0.838
I0224 00:32:05.981991   658 solver.cpp:406]     Test net output #1: loss =
0.385743 (* 1 = 0.385743 loss)
I0224 00:32:05.982347   658 solver.cpp:229] Iteration 6000, loss = 0.411537
I0224 00:32:05.982362   658 solver.cpp:245]     Train net output #0: accura
cy = 0.8
I0224 00:32:05.982373   658 solver.cpp:245]     Train net output #1: loss =
 0.411537 (* 1 = 0.411537 loss)
I0224 00:32:05.982383   658 sgd_solver.cpp:106] Iteration 6000, lr = 0.001
I0224 00:32:06.003015   658 solver.cpp:338] Iteration 7000, Testing net (#0
)
I0224 00:32:06.006283   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8388
I0224 00:32:06.006301   658 solver.cpp:406]     Test net output #1: loss =
0.384648 (* 1 = 0.384648 loss)
I0224 00:32:06.006335   658 solver.cpp:229] Iteration 7000, loss = 0.521072
I0224 00:32:06.006347   658 solver.cpp:245]     Train net output #0: accura
cy = 0.7
I0224 00:32:06.006358   658 solver.cpp:245]     Train net output #1: loss =
 0.521073 (* 1 = 0.521073 loss)
I0224 00:32:06.006368   658 sgd_solver.cpp:106] Iteration 7000, lr = 0.001
I0224 00:32:06.026715   658 solver.cpp:338] Iteration 8000, Testing net (#0
)
I0224 00:32:06.029965   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8404
I0224 00:32:06.029983   658 solver.cpp:406]     Test net output #1: loss =
0.380889 (* 1 = 0.380889 loss)
I0224 00:32:06.030015   658 solver.cpp:229] Iteration 8000, loss = 0.329477
I0224 00:32:06.030028   658 solver.cpp:245]     Train net output #0: accura
cy = 0.9
I0224 00:32:06.030040   658 solver.cpp:245]     Train net output #1: loss =
 0.329477 (* 1 = 0.329477 loss)
I0224 00:32:06.030048   658 sgd_solver.cpp:106] Iteration 8000, lr = 0.001
I0224 00:32:06.050626   658 solver.cpp:338] Iteration 9000, Testing net (#0
)
I0224 00:32:06.053889   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8376
I0224 00:32:06.053906   658 solver.cpp:406]     Test net output #1: loss =
0.382756 (* 1 = 0.382756 loss)
I0224 00:32:06.054271   658 solver.cpp:229] Iteration 9000, loss = 0.412227
I0224 00:32:06.054291   658 solver.cpp:245]     Train net output #0: accura
cy = 0.8
I0224 00:32:06.054314   658 solver.cpp:245]     Train net output #1: loss =
 0.412228 (* 1 = 0.412228 loss)
I0224 00:32:06.054337   658 sgd_solver.cpp:106] Iteration 9000, lr = 0.001
I0224 00:32:06.074646   658 solver.cpp:456] Snapshotting to binary proto fi
le examples/hdf5_classification/data/train_iter_10000.caffemodel
I0224 00:32:06.074808   658 sgd_solver.cpp:273] Snapshotting solver state t
o binary proto file examples/hdf5_classification/data/train_iter_10000.solv
erstate
I0224 00:32:06.074889   658 solver.cpp:318] Iteration 10000, loss = 0.53279
8

I0224 00:32:06.074906   658 solver.cpp:338] Iteration 10000, Testing net (#
```

```
0)
I0224 00:32:06.078208   658 solver.cpp:406]     Test net output #0: accurac
y = 0.8388
I0224 00:32:06.078225   658 solver.cpp:406]     Test net output #1: loss =
0.382042 (* 1 = 0.382042 loss)
I0224 00:32:06.078234   658 solver.cpp:323] Optimization Done.
I0224 00:32:06.078241   658 caffe.cpp:222] Optimization Done.
```

In [12]:

```
# Clean up (comment this out if you want to examine the hdf5_classification/data directo
ry).
shutil.rmtree(dirname)
```