Roadmap

❖ Evaluation Metrics
  ❖ Why metrics?
  ❖ Confusion Matrix
  ❖ IOU
  ❖ Positives vs Negatives
    ❖ ROC Curve
    ❖ Precision/Recall
    ❖ PR Curve
❖ Deep Learning & Caffe
  ❖ NN and CNN Review
  ❖ Caffe
Why metrics?

❖ Objective evaluation of an algorithm’s performance
❖ Comparing different algorithms
❖ Choosing algorithm parameters
# Confusion Matrix

## DIGIT RECOGNITION TASK

<table>
<thead>
<tr>
<th>actual</th>
<th>predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>0</td>
<td>20 0 0 6 0 0 1 0 10 0</td>
</tr>
<tr>
<td>1</td>
<td>0 25 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 31 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 21 0 0 0 0 10 0</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 31 0 0 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 0 0 22 0 0 9 0</td>
</tr>
<tr>
<td>6</td>
<td>1 0 0 1 0 2 23 0 3 1</td>
</tr>
<tr>
<td>7</td>
<td>0 8 0 0 0 0 0 23 0 0</td>
</tr>
<tr>
<td>8</td>
<td>4 0 1 3 2 1 3 0 13 4</td>
</tr>
<tr>
<td>9</td>
<td>0 0 0 2 0 0 3 1 27 0</td>
</tr>
</tbody>
</table>
IOU (Intersection over Union)
IOU (Intersection over Union)
## Positives vs Negatives

<table>
<thead>
<tr>
<th></th>
<th>Actual Positive</th>
<th>Actual Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Positive</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>Predicted Negative</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>
ROC Curve

ROC: Receiver Operating Characteristic

![ROC Curve Diagram](image)
ROC Curve

HOG + SVM threshold change
Precision and Recall

Precision: \[ \text{precision} = \frac{TP}{TP + FP} \]

Recall: \[ \text{recall} = \frac{TP}{TP + FN} \]

F-measure: \[ F = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

<table>
<thead>
<tr>
<th></th>
<th>Actual Positive</th>
<th>Actual Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted Positive</td>
<td>True Positive (TP)</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>Predicted Negative</td>
<td>False Negative (FN)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>
PR Curve
Metrics Summary

❖ Confusion Matrix
❖ IOU
❖ Positives vs Negatives
  ❖ ROC Curve, Precision-Recall Curve

❖ Related
  ❖ MAP (Mean Average Precision): PASCAL VOC + ImageNet
  ❖ Segmentation accuracy: PASCAL VOC
  ❖ Miss rate: CALTECH Pedestrian Dataset
Roadmap

❖ Evaluation Metrics
  ❖ Why metrics?
  ❖ Confusion Matrix
  ❖ IOU
  ❖ Positives vs Negatives
    ❖ ROC Curve
    ❖ Precision/Recall
    ❖ PR Curve
❖ Deep Learning & Caffe
  ❖ NN and CNN Review
  ❖ Caffe
Neural Networks

\[ x \]

\[ y = W_2 h \]

\[ h = W_1 x \]

“Fully-connected” layers (matrix multiplication)

courtesy Andrej Karpathy & Justin Johnson
Convolutional Neural Networks
Convolution Layer

32x32x3 image
5x5x3 filter
convolve (slide) over all spatial locations
activation map
Activation Functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**tanh**  \[ \tanh(x) \]

**ReLU**  \[ \max(0, x) \]

**Leaky ReLU**  \[ \max(0.1x, x) \]

**Maxout**  \[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \]
Pooling Layer

- Input: 224x224x64
- Pooling: reduces to 112x112x64
- Downsampling: further reduces to 112x112
Classification layer

- Softmax function
- Output between 0 and 1 for each class

\[ f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \]
Convolutional Neural Networks
Back-propagation

Stochastic Gradient Descent (SGD)

\[ w \leftarrow w - \alpha \frac{\partial E}{\partial w} \]

- SGD with momentum
- Nesterov momentum
- Adagrad
- RMSProp
Caffe

- Deep Learning Framework from Berkeley
- C++, Python, MATLAB
- Available at https://github.com/BVLC/caffe
- CPU and GPU mode
- Platform-specific installation instructions available at http://caffe.berkeleyvision.org/installation.html
Blob: Storage and Communication of Data

- Data blobs are $N \times C \times H \times W$

Net: Contains all the layers in the networks

- Performs forward/backward pass through the entire network

Solver: Used to set training/testing parameters

- Number of iterations, back propagation method, etc.
Network Definition(train.prototxt)

```protobuf
data_param {
  source: "examples/mnist/mnist_train_lmdb"
  batch_size: 64
  backend: LMDB
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

```protobuf
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
}
```
Network Definition (train.prototxt)

layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "pool2"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}

layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
Network Definition (train.prototxt)

```protobuf
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```
Network Definition (test.prototxt)

Previously

name: "LeNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_train_lmdb"
    batch_size: 64
    backend: LMDB
  }
}

layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 100
    backend: LMDB
  }
}
Network Definition (test.prototxt)

Previously

```plaintext
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

```plaintext
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
```
# The train/test net protocol buffer definition

net: "examples/mnist/lenet_train_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 and 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: "step"

gamma: 0.1

steps: 3000

# Display every 100 iterations
display: 100

# The maximum number of iterations
max_iter: 10000

# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"

# solver mode: CPU or GPU

solver_mode: GPU
Training from the Command Line

./build/tools/caffe train -solver examples/mnist/lenet_solver.prototxt
OR
./build/tools/caffe train -solver examples/mnist/lenet_solver.prototxt
   -snapshot examples/mnist/lenet_iter_5000.solverstate

Command line output

I1203 net.cpp:66] Creating Layer conv1
I1203 net.cpp:76] conv1 <- data
I1203 net.cpp:101] conv1 -> conv1
I1203 net.cpp:116] Top shape: 20 24 24
I1203 net.cpp:127] conv1 needs backward computation.

I1203 solver.cpp:204] Iteration 100, lr = 0.00992565
I1203 solver.cpp:66] Iteration 100, loss = 0.26044
...
I1203 solver.cpp:84] Testing net
I1203 solver.cpp:111] Test score #0: 0.9785
I1203 solver.cpp:111] Test score #1: 0.0606671
Testing from the command line

```
./build/tools/caffe test -model examples/mnist/lenet_train_test.prototxt
-weights examples/mnist/lenet_iter_10000.caffemodel -gpu 0 -iterations 100
```
PyCaffe (Training in Python)

- Add caffe python directory to path and import caffe

```python
caffe_root = '../' # this file should be run from {caffe_root}/examples (otherwise change this line)

import sys
sys.path.insert(0, caffe_root + 'python')
import caffe
```
Use NetSpec to define layers

```python
from caffe import layers as L, 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.fc1 = L.InnerProduct(n.pool2, num_output=500, weight_filler=dict(type='xavier'))
    n.relu1 = L.ReLU(n.fc1, in_place=True)
    n.score = L.InnerProduct(n.relu1, num_output=10, weight_filler=dict(type='xavier'))
    n.loss = L.SoftmaxWithLoss(n.score, n.label)

    return n.to_proto()

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

with open('mnist/lenet_auto_test.prototxt', 'w') as f:
    f.write(str(lenet('mnist/mnist_test_lmdb', 100)))
```
Define solver and train network

```python
caffe.set_device(0)
caffe.set_mode_gpu()

### load the solver and create train and test nets
solver = None  # ignore this workaround for lmdb data (can't instantiate two solvers on the same data)
solver = caffe.SGDSolver('mnist/lenet_auto_solver.prototxt')
```

```python
[(k, v.data.shape) for k, v in solver.net.blobs.items()]
```

```python
(('data', (64, 1, 28, 28)),
 ('label', (64,)),
 ('conv1', (64, 20, 24, 24)),
 ('pool1', (64, 20, 12, 12)),
 ('conv2', (64, 50, 8, 8)),
 ('pool2', (64, 50, 4, 4)),
 ('fc1', (64, 500)),
 ('score', (64, 10)),
 ('loss', ()))
```

```python
solver.net.forward()  # train net
```
Access Net data

# we use a little trick to tile the first eight images
imshow(solver.net.blobs['data'].data[8, 0].transpose(1, 0, 2).reshape(28, 8*28), cmap='gray'); axis('off')
print 'train labels:', solver.net.blobs['label'].data[8]

train labels: [ 5.  0.  4.  1.  9.  2.  1.  3.]

50419213
# load the model
net = caffe.Net('models/bvlc_reference_caffenet/train.prototxt',
                'models/bvlc_reference_caffenet/train_iter30000.caffemodel',
                caffe.TEST)

# load input and configure preprocessing
transformer = caffe.io.Transformer({'data': net.blobs['data'].data.shape})
transformer.set_mean('data', np.load('ilsvrc_2012_mean.npy').mean(1).mean(1))
transformer.set_transpose('data', (2,0,1))
transformer.set_channel_swap('data', (2,1,0))
transformer.set_raw_scale('data', 255.0)

# note we can change the batch size on-the-fly
# since we classify only one image, we change batch size from 10 to 1
net.blobs['data'].reshape(1,3,227,227)

# load the image in the data layer
im = caffe.io.load_image('examples/images/cat.jpg')
net.blobs['data'].data[:] = transformer.preprocess('data', im)

# compute
out = net.forward()

# other possibility : out = net.forward_all(data=np.asarray([transformer.preprocess('data', im)]))

# predicted predicted class
print out['prob'].argmax()
Caffe Summary

- Install Caffe
- Define training network in train.prototxt
- Define test network in test.prototxt
- Define solver parameters in solver.prototxt
- Train and test
CNN Training tips

❖ Before running final/long training
❖ Make sure you can overfit on a small training set
❖ Make sure your loss decreases over first several iterations
❖ Otherwise adjust parameter until it does, especially learning rate
❖ Separate train/val/test data
References

❖ Metrics
  ❖ http://vase.essex.ac.uk/talks/performance-evaluation.pdf

❖ CNN
  ❖ http://cs231n.github.io/

❖ Caffe
  ❖ http://caffe.berkeleyvision.org/installation.html
  ❖ https://github.com/BVLC/caffe/tree/master/examples
  ❖ http://caffe.berkeleyvision.org/tutorial/interfaces.html
QUESTIONS?