Day 1 Lecture 6

Software Frameworks for Deep Learning

Packages

- Caffe
  - NVIDIA Digits
- Theano
  - Lasagne
  - Keras
  - Blocks
- Torch
- TensorFlow
- MxNet
- MatConvNet
- Nervana Neon
- Leaf

Caffe

Deep learning framework from Berkeley (BVLC)

- http://caffe.berkeleyvision.org/
- Implemented in C++
- CPU and GPU modes (CUDA)
- Python wrapper
- Command line tools for training and prediction
- Uses **Google protobuf** based model specification and parameter format
- Several supported data formats (file system, leveldb, lmdb, hdf5)

Caffe

```
name: "AlexNet"
layer {
 name: "data"
 type: "Input"
 top: "data"
 input_param { shape: { dim: 10 dim: 3 dim: 227 dim: 227 } }
}
layer {
 name: "conv1"
 type: "Convolution"
 bottom: "data"
 top: "conv1"
 param { lr_mult: 1  decay_mult: 1 }
 param { lr_mult: 2  decay_mult: 0 }
 convolution_param {
   num_output: 96  kernel_size: 11  stride: 4 }
}
layer { name: "relu1” type: "ReLU"
   bottom: "conv1" top: "conv1" }
```

```
net: "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/my_model"
```

```
$ ./build/tools/caffe train \\
      --solver=solver.prototxt
```

Caffe

Pros

- Portable models
- Declarative model spec
- Simple command line interface for training and fine tuning
- Fast and fairly small memory footprint (relatively)
- Python layers

Cons

- Lots of dependencies; can be tricky to install
- No automatic differentiation
- Not so convenient to extend (write layers in C++ or Python, handwritten CUDA code)
- Less flexible that some other frameworks
- Python interface does not expose everything

NVIDIA Digits

Web based UI that sits on top of Caffe

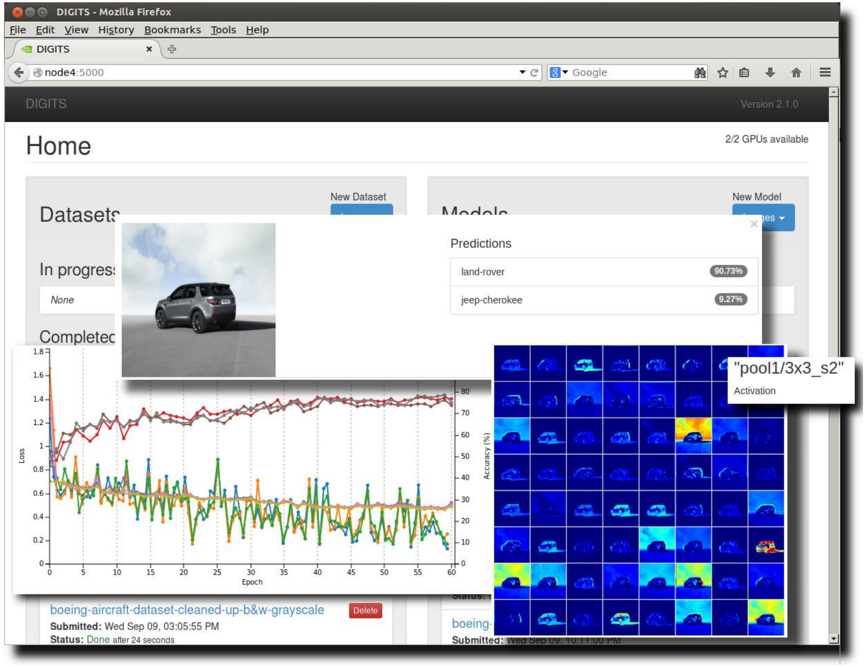
Create datasets

Train and test models

Visualize learning curves

Visualize layer outputs and predictions

https://developer.nvidia.com/digits



Theano

Define, evaluate, optimize mathematical expressions in Python

- http://deeplearning.net/software/theano/
- Symbol graph based approach
- Can be used for lots more than just deep learning
- Automatic differentiation
- Fairly low-level API (define layers yourself, or use Lasagne/Blocks/Keras)
- Very flexible and customizable
- Execute on CPU or GPU

Theano

theano

Pros

- Python
- Super flexible
- Automatic differentiation
- CUDA support
- Tight numpy integration

Cons

- Slow graph compile times
- Low-level API

Lasagne

http://lasagne.readthedocs.org/en/latest/

```
import lasagne
import theano
import theano.tensor as T

# create Theano variables for input and target minibatch
input_var, target_var = T.tensor4('X'), T.ivector('y')

# create a small convolutional neural network
from lasagne.nonlinearities import leaky_rectify, softmax
network = lasagne.layers.InputLayer((None, 3, 32, 32), input_var)
network = lasagne.layers.Conv2DLayer(network, 64, (3, 3),
                                      nonlinearity=leaky_rectify)
network = lasagne.layers.Conv2DLayer(network, 32, (3, 3),
                                      nonlinearity=leaky_rectify)
network = lasagne.layers.Pool2DLayer(network, (3, 3), stride=2, mode='max')
network = lasagne.layers.DenseLayer(lasagne.layers.dropout(network, 0.5),
                                    128, nonlinearity=leaky_rectify,
                                    W=lasagne.init.Orthogonal())
network = lasagne.layers.DenseLayer(lasagne.layers.dropout(network, 0.5),
                                    10, nonlinearity=softmax)
```

Lasagne

```
# create loss function
prediction = lasagne.layers.get_output(network)
loss = lasagne.objectives.categorical_crossentropy(prediction, target_var)
loss = loss.mean() + 1e-4 * lasagne.regularization.regularize_network_params(
        network, lasagne.regularization.l2)

# create parameter update expressions
params = lasagne.layers.get_all_params(network, trainable=True)
updates = lasagne.updates.nesterov_momentum(loss, params, learning_rate=0.01, momentum=0.9)

# compile training function that updates parameters and returns training loss
train_fn = theano.function([input_var, target_var], loss, updates=updates)

# train network (assuming you've got some training data in numpy arrays)
for epoch in range(100):
    loss = 0
    for input_batch, target_batch in training_data:
        loss += train_fn(input_batch, target_batch)
    print("Epoch %d: Loss %g" % (epoch + 1, loss / len(training_data)))
```

Lasagne

Pros

- Python
- Simple: easy to use layers
- Transparent: thin layer over theano - can do everything theano can do
- Flexible: easy to create custom layers

Cons

- Slow graph compile times

Keras

- Also built on Theano (has a TensorFlow backend now too)
- Simple Torch-like model spec API
    - Easy to specify sequential models
- Scikit-learn style fit/predict functions
- Different design philosophy to Lasagne: hides Theano implementation

```python
from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))
```

```python
model.compile(loss='categorical_crossentropy',
      optimizer='sgd', metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5,  batch_size=32)

loss_and_metrics = model.evaluate(
      X_test, Y_test, batch_size=32)

classes = model.predict_classes(X_test, batch_size=32)
proba = model.predict_proba(X_test, batch_size=32)
```

Torch

Scientific computing framework for Lua

- http://torch.ch/
- Very fast (LuaJIT)
- Flexible
- Used by Facebook, Deepmind, Twitter

Cons

- No automatic differentiation built-in (Twitter autograd implements this)
- No Python “batteries included”

```
net = nn.Sequential()
net:add(nn.SpatialConvolution(1, 6, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.SpatialConvolution(6, 16, 5, 5))
net:add(nn.ReLU())
net:add(nn.SpatialMaxPooling(2,2,2,2))
net:add(nn.View(16*5*5))
net:add(nn.Linear(16*5*5, 120))
net:add(nn.ReLU())
net:add(nn.Linear(120, 84))
net:add(nn.ReLU())
net:add(nn.Linear(84, 10))
net:add(nn.LogSoftMax())

output = net:forward(input)
```

TensorFlow

Google's new deep learning library

- https://www.tensorflow.org/
- Similar to Theano: symbolic computing graph approach
- C++ with first class Python bindings
- Distributed computing support (since April 13, 2016)
- Good documentation
- Flexible
- No graph compilation step needed
- Early versions were slow in benchmarks (now resolved!)
- Memory issues in earlier versions

TensorFlow example

```
import tensorflow as tf

sess = tf.InteractiveSession()

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  train_step.run({x: batch_xs, y_: batch_ys})
```

TensorFlow Slim

Lightweight library for defining, training, and evaluating models in TensorFlow

Enables defining complex networks quickly and concisely

Less boilerplate!

```python
def vgg16(inputs):
  with slim.arg_scope([slim.ops.conv2d, slim.ops.fc], stddev=0.01, weight_decay=0.0005):
    net = slim.ops.repeat_op(2, inputs, slim.ops.conv2d, 64, [3, 3], scope='conv1')
    net = slim.ops.max_pool(net, [2, 2], scope='pool1')
    net = slim.ops.repeat_op(2, net, slim.ops.conv2d, 128, [3, 3], scope='conv2')
    net = slim.ops.max_pool(net, [2, 2], scope='pool2')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 256, [3, 3], scope='conv3')
    net = slim.ops.max_pool(net, [2, 2], scope='pool3')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 512, [3, 3], scope='conv4')
    net = slim.ops.max_pool(net, [2, 2], scope='pool4')
    net = slim.ops.repeat_op(3, net, slim.ops.conv2d, 512, [3, 3], scope='conv5')
    net = slim.ops.max_pool(net, [2, 2], scope='pool5')
    net = slim.ops.flatten(net, scope='flatten5')
    net = slim.ops.fc(net, 4096, scope='fc6')
    net = slim.ops.dropout(net, 0.5, scope='dropout6')
    net = slim.ops.fc(net, 4096, scope='fc7')
    net = slim.ops.dropout(net, 0.5, scope='dropout7')
    net = slim.ops.fc(net, 1000, activation=None, scope='fc8')
  return net
```

Other deep learning libraries

MxNet

- https://mxnet.readthedocs.org/en/latest/index.html
- Relative newcomer, under active development
- Blazingly fast
- Distributed computing support
- Bindings for C++, Python, R, Scala, Julia, MATLAB, and Javascript

MatConvNet

- http://www.vlfeat.org/matconvnet/
- MATLAB toolbox for CNNs

Nervana Neon

- http://neon.nervanasys.com/docs/latest/index.html
- Blazingly fast
- Commercial, but open source
- 16-bit floating point support

AutumnAI Leaf

- http://autumnai.com/
- Rust-based toolkit
- Performance similar to Torch

	Speed	Memory	Distributed	Languages	Flexibility	Simplicity
Caffe	XXX	XXX	No	C++/Python	X	XX
Theano	XX		No	Python	XXXX	X
Lasagne	XX		No	Python	XXXX	XXX
Keras	XX		No	Python	XX	XXXX
Torch	XXXX		No	Lua	XXXX	XXX
TensorFlow	XXX		Yes	C++/Python	XXXX	XX
MxNet	XXXX	XXX	Yes	Python, Julia, R, MATLAB ...	XXX	XX
MatConvNet	XX		No	MATLAB	XX	XXX
Neon	XXXXX		No	Python	XX	XXX
Leaf	XXXX	XXX	No	Rust	?	?

cuDNN

- New versions of cuDNN have somewhat leveled the playing field in terms of performance on the GPU
    - Memory
    - Speed
    - Throughput
- All major deep learning libraries can use it
    - Torch
    - Theano (Keras, Lasagne)
    - TensorFlow
- Choice of framework now is largely a matter of taste
    - Preferred language
    - Ease of use
    - Flexibility