theano

http://www.deeplearning.net/software/theano

http://groups.google.com/group/theano-announce
Theano: A CPU and GPU Math Expression Compiler

James Bergstra, Olivier Breuleux, Frederic Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, Yoshua Bengio
1: `import` theano
2: `a = theano.tensor.vector('a')`  # declare variable
3: `b = a + a**10`  # build expression
4: `f = theano.function([a, b])`  # compile function
5: `print `f([0,1,2])`'  # call function
6: `# prints `array([0,2,1026])`'
1: `import theano`
2: `a = theano.tensor.vector('a')`  # declare variable
3: `b = a + a**10`  # build expression
4: `f = theano.function([a, b])`  # compile function
5: `print 'f([0,1,2])'`  # call function
6: `# prints 'array([0,2,1026])'`
1: `import` `theano`
2: `a = theano.tensor.vector('a')`  # declare variable
3: `b = a + a**10`  # build expression
4: `f = theano.function([a], b)`  # compile function
5: `print 'f([0,1,2])'`  # call function
6: `# prints 'array([0,2,1026])'`
7: `No pow() ??`
A real example: Logistic Regression

- GPU-ready
- symbolic differentiation
- speed optimizations
- stability optimizations
1: import numpy
2: import theano.tensor as T
3: from theano import shared, function
4: rng = numpy.random
5: 
6: # Declare Theano variables
7: x = T.matrix()
8: y = T.vector()
9: w = shared(rng.randn(100))
10: b = shared(numpy.zeros(0))
11: print "Initial model:
12: print w.value, b.value
13: 
14: # Construct Theano expression graph
15: p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
16: xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
17: prediction = p_1 > 0.5
18: cost = xent.mean() + 0.01*(w**2).sum()
19: gw, gb = T.grad(cost, [w, b])
20: 
21: # Compile expressions to functions
22: train = function(  
23:     inputs=[x, y],  
24:     outputs=[prediction, xent],  
25:     updates={w:w-0.1*gw, b:b-0.1*gb})
26: predict = function(inputs=[x], outputs=prediction)
1: import numpy
2: import theano.tensor as T
3: from theano import shared, function
4: rng = numpy.random
5: 
6: # Declare Theano variables
7: x = T.matrix()
8: y = T.vector()
9: w = shared(rng.randn(100))
10: b = shared(numpy.zeros())
11: print "Initial model:",
12: print w.value, b.value
13: 
14: # Construct Theano expression graph
15: p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
16: xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
17: prediction = p_1 > 0.5
18: cost = xent.mean() + 0.01*(w**2).sum()
19: gw,gb = T.grad(cost, [w,b])
20: 
21: # Compile expressions to functions
22: train = function(
23:     inputs=[x,y],
24:     outputs=[prediction, xent],
25:     updates={w:w-0.1*gw, b:b-0.1*gb})
26: predict = function(inputs=[x], outputs=prediction)
1: `import` numpy
2: `import` theano.tensor as T
3: `from` theano `import` shared, function
4: `rng = numpy.random`
5: 
6: `# Declare Theano variables`
7: `x = T.matrix()`
8: `y = T.vector()`
9: `w = shared(rng.randn(100))`
10: `b = shared(numpy.zeros())`
11: `print "Initial model:"
12: `print w.value, b.value`
13: 
14: `# Construct Theano expression graph`
15: `p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))`
16: `xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)`
17: `prediction = p_1 > 0.5`
18: `cost = xent.mean() + 0.01*(w**2).sum()`
19: `gw, gb = T.grad(cost, [w, b])`
20: 
21: `# Compile expressions to functions`
22: `train = function`
23: `       inputs=[x, y],`
24: `       outputs=[prediction, xent],`
25: `       updates={w:w-0.1*gw, b:b-0.1*gb})`
26: `predict = function(inputs=[x], outputs=prediction)`
```python
1: import numpy
2: import theano.tensor as T
3: from theano import shared, function
4: rng = numpy.random
5: 
6: # Declare Theano variables
7: x = T.matrix()
8: y = T.vector()
9: w = shared(rng.randn(100))
10: b = shared(numpy.zeros())
11: print "Initial model:"
12: print w.value, b.value
13: 
14: # Construct Theano expression graph
15: p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
16: xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
17: prediction = p_1 > 0.5
18: cost = xent.mean() + 0.01*(w**2).sum()
19: gw, gb = T.grad(cost, [w, b])
20: 
21: # Compile expressions to functions
22: train = function(
23:     inputs=[x, y],
24:     outputs=[prediction, xent],
25:     updates={w:w-0.1*gw, b:b-0.1*gb})
26: predict = function(inputs=[x], outputs=prediction)
27:
```
Loop Fusion

logistic sigmoid

log1p(exp(x))

14: # Construct Theano expression graph
15: p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
16: xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
17: prediction = p_1 > 0.5
18: cost = xent.mean() + 0.01*(w**2).sum()
19: gw, gb = T.grad(cost, [w, b])
20:
21: # Compile expressions to functions
22: train = function(
23:     inputs=[x, y],
24:     outputs=[prediction, xent],
25:     updates={w:w-0.1*gw, b:b-0.1*gb})
26: predict = function(inputs=[x], outputs=prediction)
Some Benchmarks

- Multi-layer Perceptron
- Convolutional Neural Network
- Misc. Elementwise

Competitors

- NumPy+SciPy, MATLAB
- EBLearn, Torch5
- numexpr
Multi-Layer Perceptron: 60x784 matrix times 784x500 matrix, tanh, times 500x10 matrix, elemwise, then all in reverse for backpropagation.
Convolutional Network: 256x256 images convolved with 6 7x7 filters, downsampling to 6x50x50, tanh, convolution with 16 6x7x7 filters, tanh, matrix multiply, elemwise, then all in reverse for backpropagation.
Solid blue: Theano
Dashed red: `numexpr` (without MKL)
Theano in Summary

- High Level functional description of calculation
- Rearranges expressions for speed and stability
- Compiles many expressions to machine language
- Uses GPU implementations (where possible)

http://www.deeplearning.net/software/theano

http://groups.google.com/group/theano-announce
Theano Sprint

Ideas:

- Get Theano running on your computer
- Get an algorithm running with Theano
- Improve PyCUDA integration?
- Translate to/from SymPy graphs?
- Improve linear algebra support?
- Implement a conjugate solver?
- Advanced Indexing with gradient??