GPU computing for machine learning with theano

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

Presenting: Olivier Grisel

Authors:
James Bergstra, Olivier Breuleux,
Frederic Bastien, Pascal Lamblin,
Razvan Pascanu, Guillaume Desjardins,
Joseph Turian, Yoshua Bengio

Université de Montréal

A CPU and GPU Math Expression Compiler

SymPy

cython

pycuda

threading

theano

pyopencl

multiprocessing

numpy

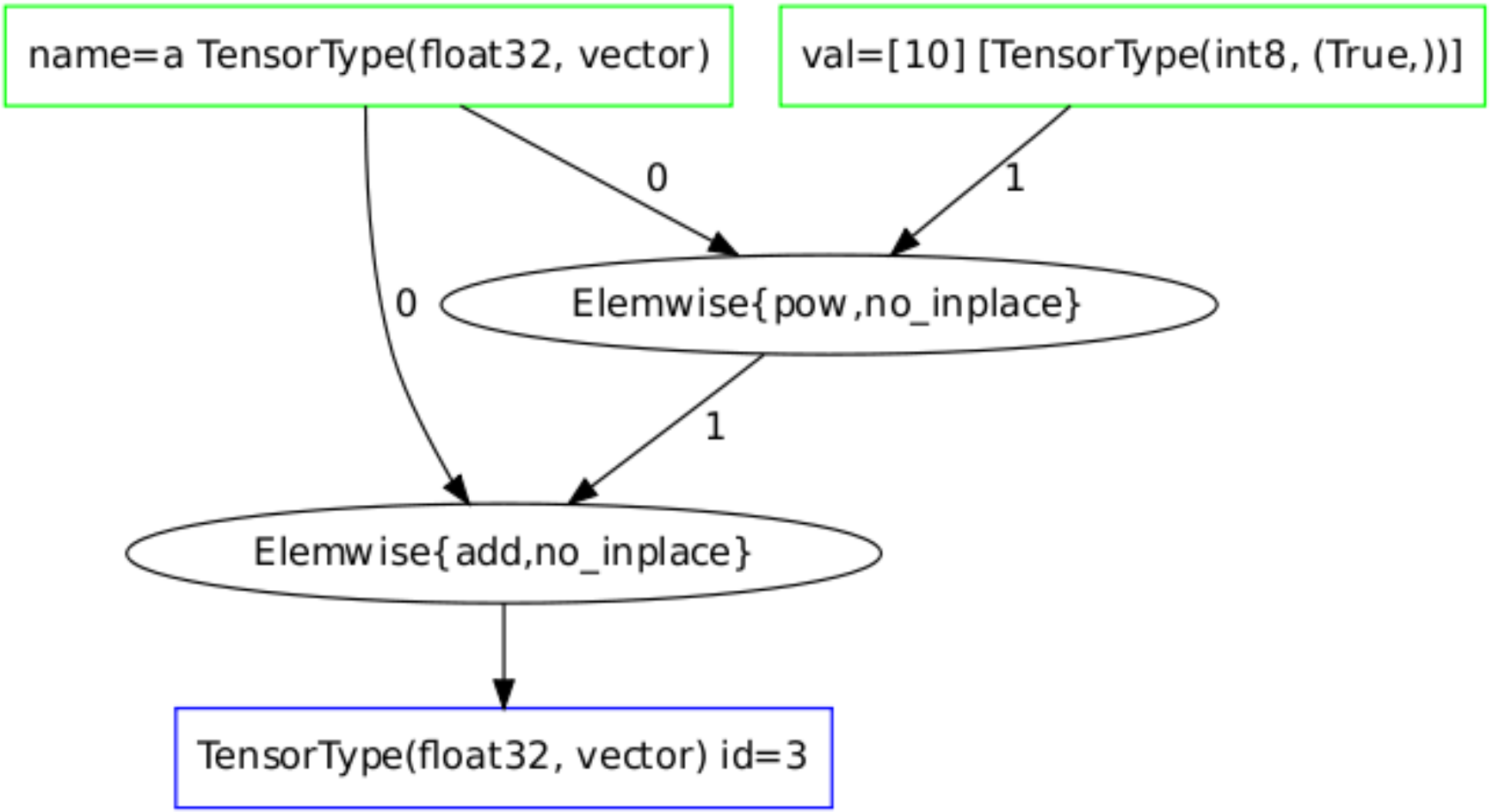
numexpr

scipy.linalg

scipy.weave

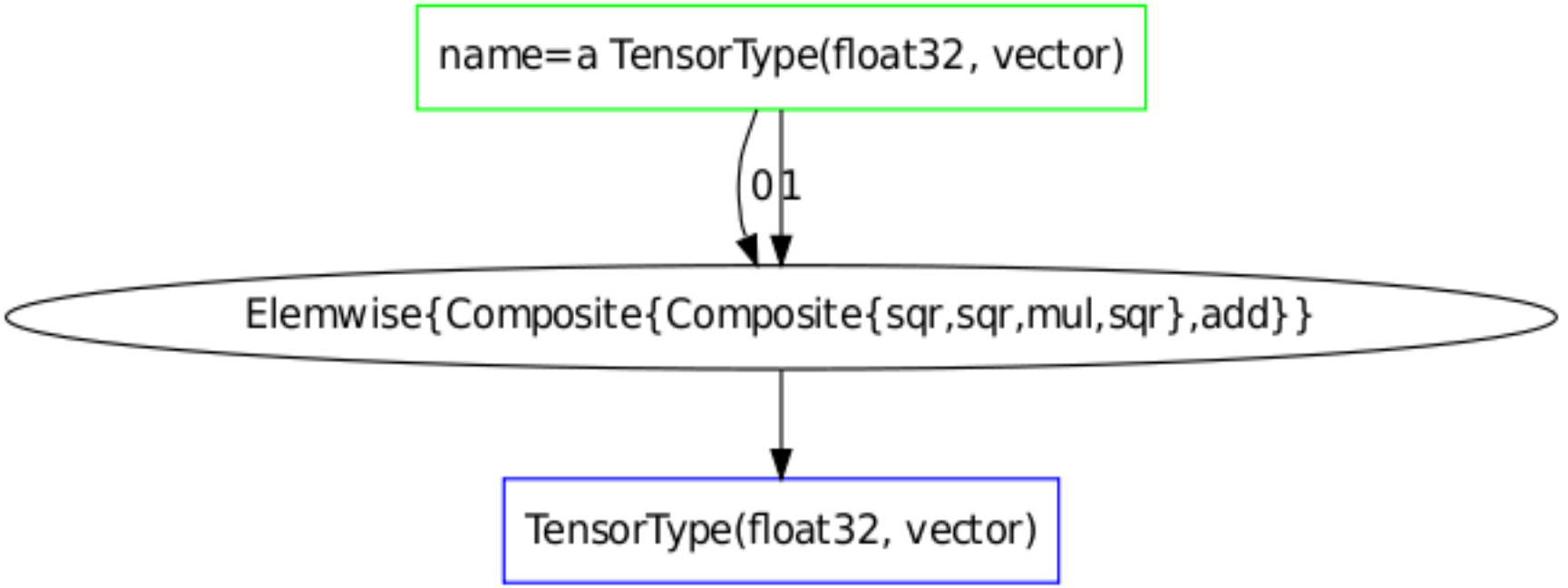
scipy.signal

```
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:
7:
```




```
theano.printing.pydotprint(
    theano.function([a], b, mode="FAST_COMPILE"))
```

```
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:
7:
```



theano.printing.pydotprint(f)

A real example: Logistic Regression

- GPU -ready
- Symbolic differentiation
- Speed optimizations
- Stability optimizations

```
import numpy
import theano.tensor as T
from theano import shared, function
rng = numpy.random

# Declare Theano variables
x = T.matrix()
y = T.vector()
w = shared(rng.randn(100))
b = shared(numpy.zeros(()))
print "Initial model:"
print w.value, b.value

# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
prediction = p_1 > 0.5
cost = xent.mean() + 0.01*(w**2).sum()
gw,gb = T.grad(cost, [w,b])

# Compile expressions to functions
train = function(
            inputs=[x,y],
            outputs=[prediction, xent],
            updates={w:w-0.1*gw, b:b-0.1*gb})
predict = function(inputs=[x], outputs=prediction)
```

```
N = 4
feats = 100
D = (rng.randn(N, feats), rng.randint(size=4,low=0, high=2))
training_steps = 10
for i in range(training_steps):
    pred, err = train(D[0], D[1])
print "Final model:"
print w.value, b.value

print "target values for D"
print D[1]

print "prediction on D"
print predict(D[0])
```

Loop Fusion

log(1+x) => log1p(x)

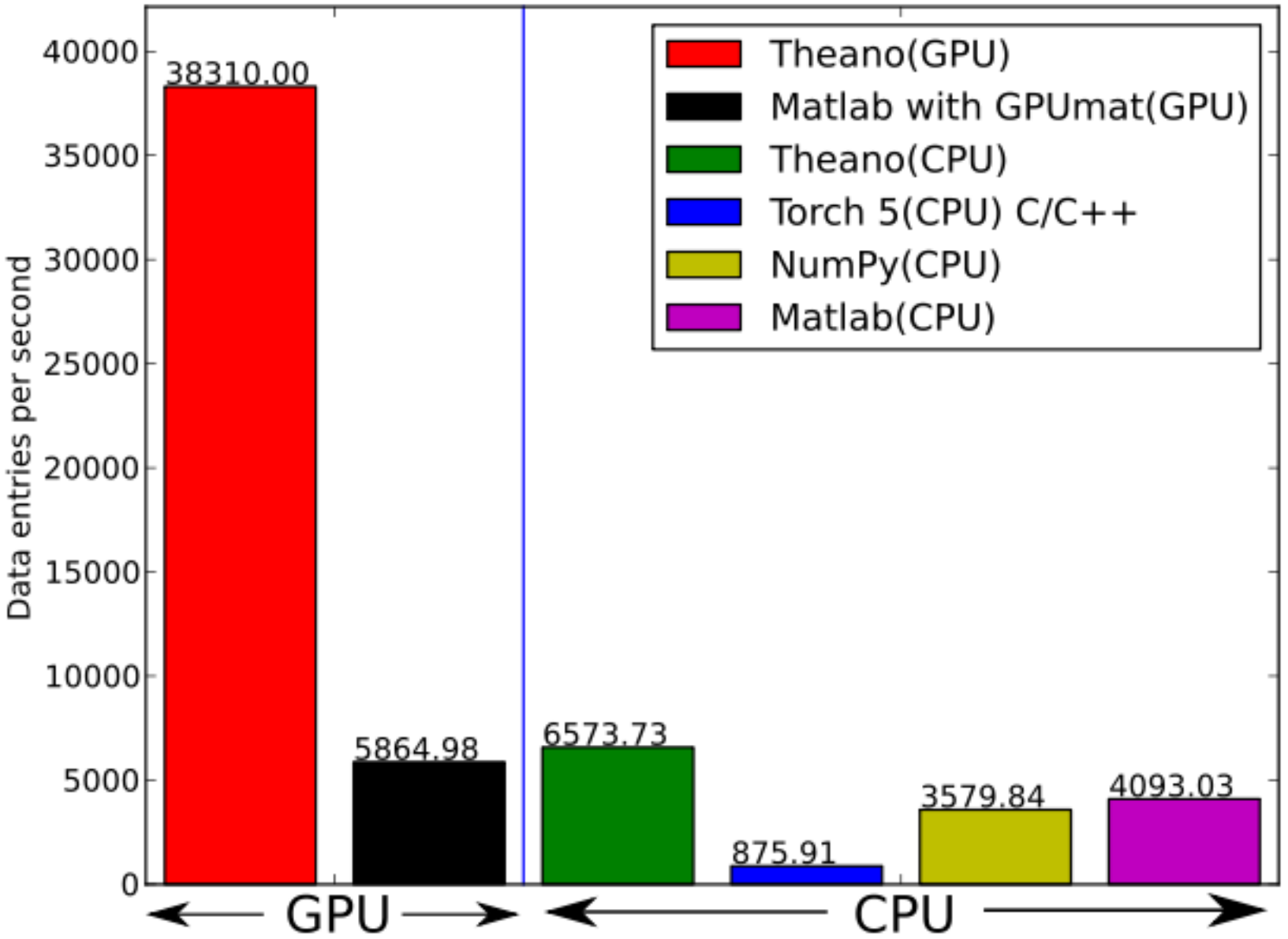
Logistic Sigmoid => atanh

a*M.x + b.y => blas DGMEV

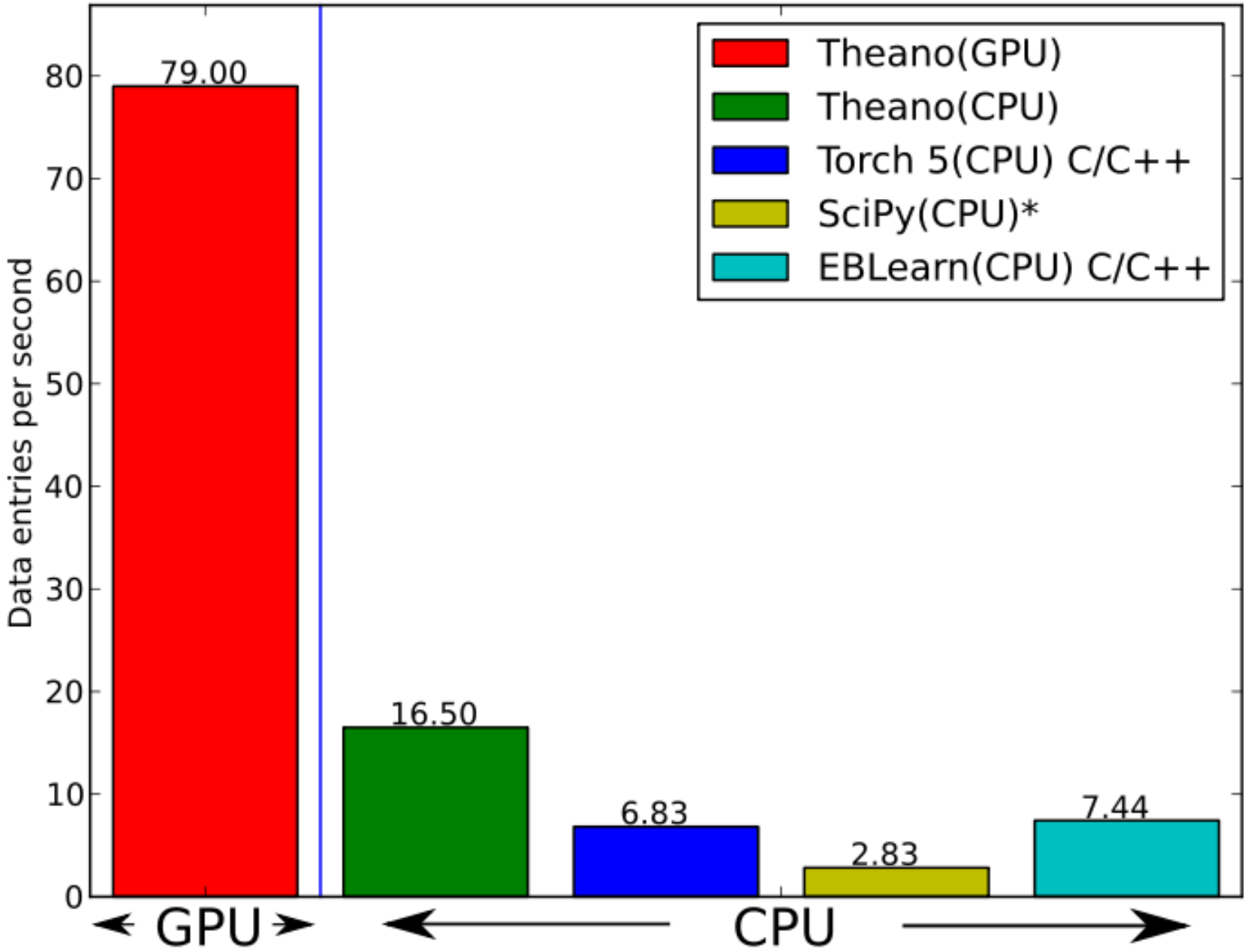
```
# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w)-b))
xent = -y*T.log(p_1) - (1-y)*T.log(1-p_1)
prediction = p_1 > 0.5
cost = xent.mean() + 0.01*(w**2).sum()
gw,gb = T.grad(cost, [w,b])

# Compile expressions to functions
train = function(
            inputs=[x,y],
            outputs=[prediction, xent],
            updates={w:w-0.1*gw, b:b-0.1*gb})
predict = function(inputs=[x], outputs=prediction)
```

Some Benchmarks

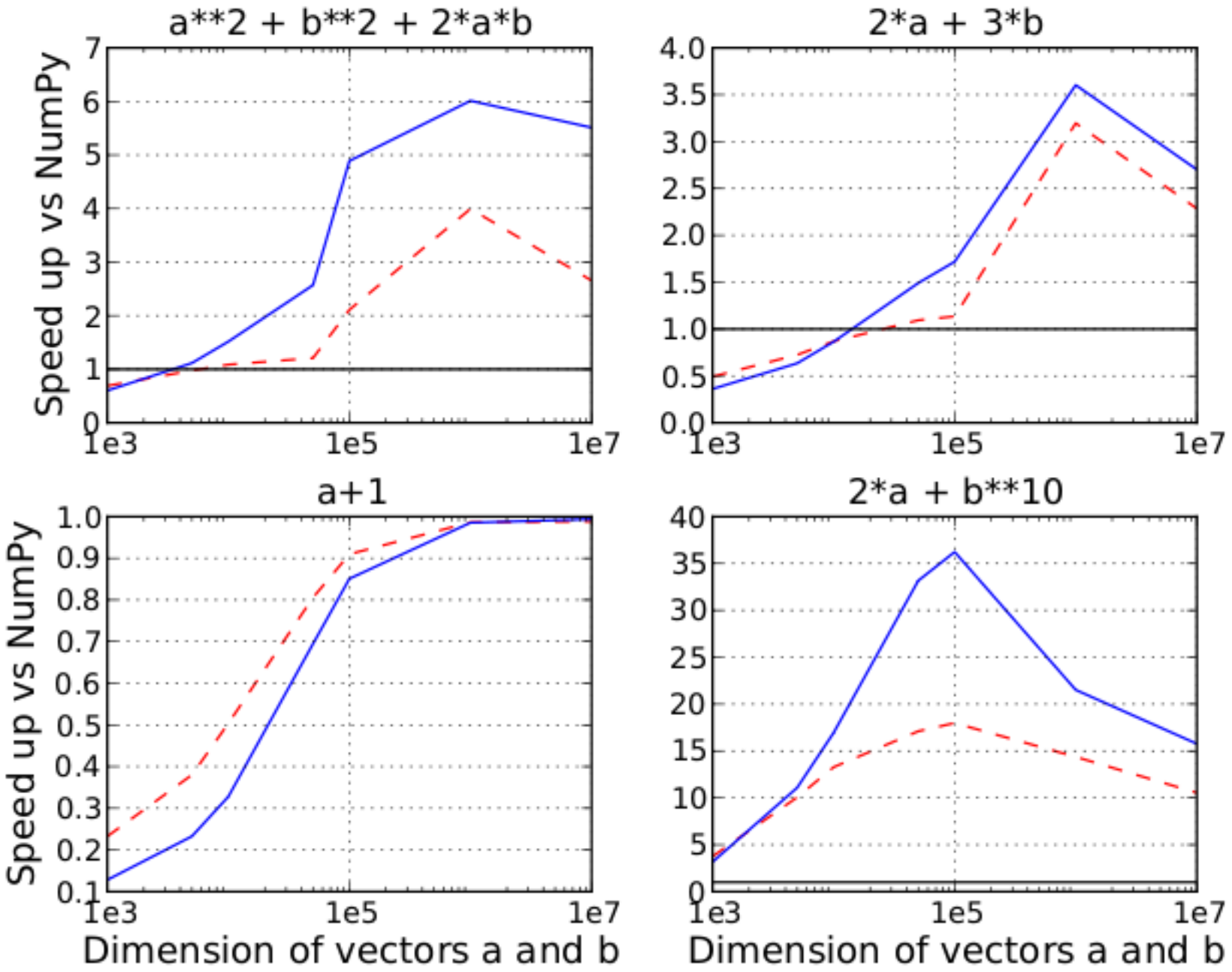
- Sample machine learning programs
  - Multi-layer Perceptron
  - Convolutional Neural Network
  - Misc. Elementwise
- Typical machine learning libs
  - NumPy+SciPy, MATLAB
  - EBLearn, Torch5 (C / C++ libs)
  - numexpr



MLP 60x784 matrix times 784x500 matrix, tanh, times 500x10 matrix + same with backprop



Convolutional Network: 256x256 images convolved with 6 7x7 filters, downsampling to 6x50x50, tanh, convolution with 16 6x7x7 filters, tanh, matrix multiply + same with backprop



- blue: theano CPU - red: numexpr without MKL

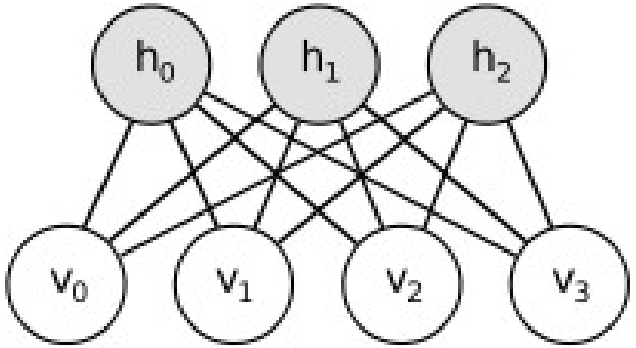
Applications to machine learning

Best tutorial ever:

http://deeplearning.net/tutorial

Logistic Regression, MLP, Convolutional Neural Networks, Stacked Denoising Autoencoders, Restricted Boltzmann Machines

Restricted Boltzmann Machines

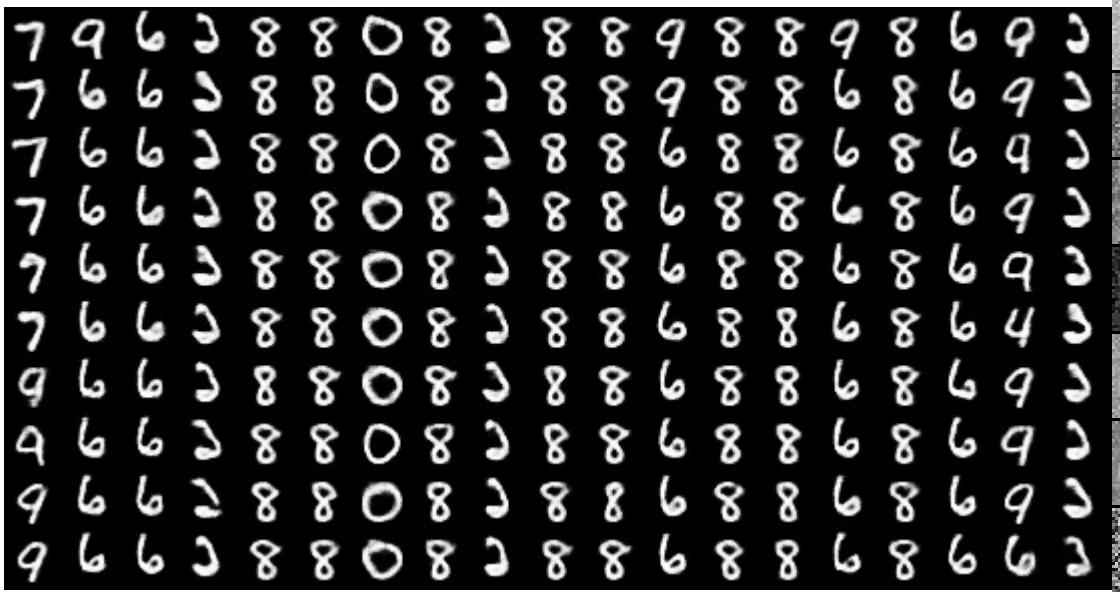


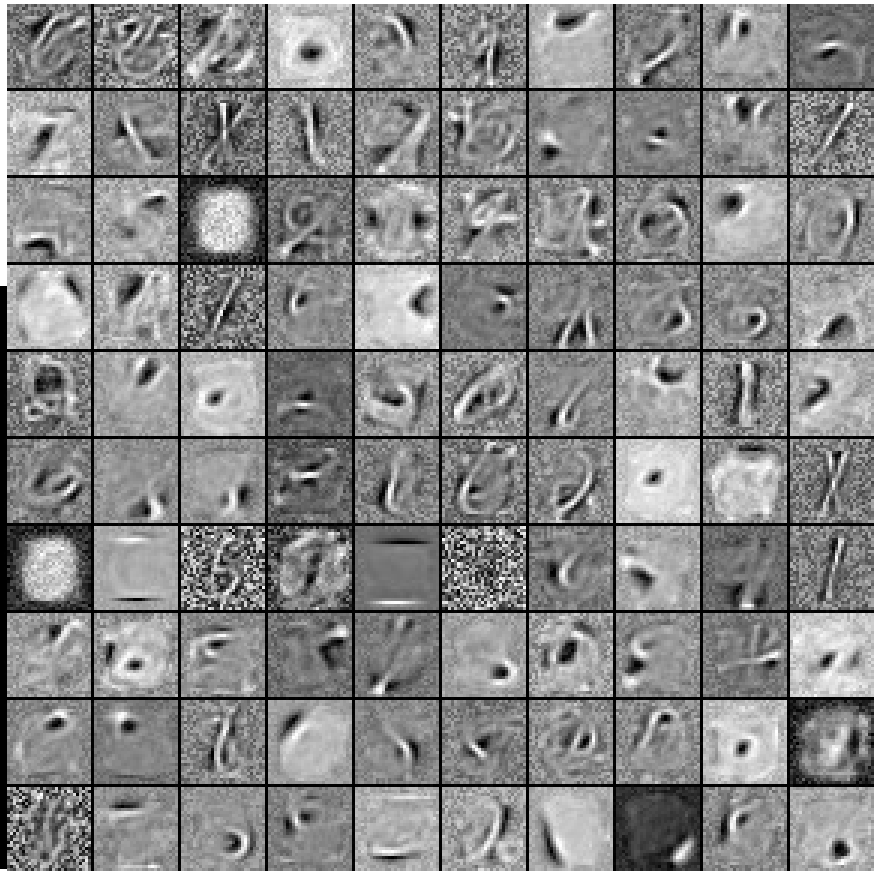
$$E(v,h) = -b'v - c'h - h'Wv$$

$$P(x) = \sum_h P(x,h) = \sum_h \frac{e^{-E(x,h)}}{Z}.$$

$$P(h_i = 1|v) = sigm(c_i + W_i v)$$

$$P(v_j = 1|h) = sigm(b_j + W'_j h)$$





Main theano take-aways

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

http://deeplearning.net/software/theano

Fork & Follow

http://{github,twitter}.com/ogrisel