Dataflow Programming: Theano Example

```
$ python
Python 2.7.6
>>> import theano
Using gpu device 0: GeForce GTX TITAN X (CNMeM is disabled, cuDNN 5005)
>>> a = theano.tensor.scalar()  # symbolic variable
>>> b = theano.tensor.scalar()  # symbolic variable
>>> c = a+b   # define computation graph - this line is executed only once!
>>> f = theano.function([a,b], c)  # compile fcn. that maps [a,b] to c
>>> f(1,1000)
array(1001.0, dtype=float32)
>>> f(2,2)
array(4.0, dtype=float32)
>>> f(1,2)   # execute f as often as you want
array(3.0, dtype=float32)

>>> type(a)
<class 'theano.tensor.var.TensorVariable'>
>>> type(b)
<class 'theano.tensor.var.TensorVariable'>
>>> type(c)
<class 'theano.tensor.var.TensorVariable'>
```

Neural Networks in Lasagne

```
inputs  = theano.tensor.tensor4()  # 4-dimensional symbolic array
targets = theano.tensor.tensor4()  # 4-dimensional symbolic array
nchannels = 3
network = lasagne.layers.InputLayer(shape=(None, nchannels, None, None), input_var=inputs)
network = lasagne.layers.Conv2DLayer(network, num_filters=128, filter_size=(3,3))
network = lasagne.layers.Pool2DLayer(network, pool_size=(2,2))
network = lasagne.layers.Conv2DLayer(network, num_filters=256, filter_size=(3,3))
network = lasagne.layers.GlobalPoolLayer(network, pool_function=theano.tensor.max)
network = lasagne.layers.DenseLayer(network, num_units=1024)
network = lasagne.layers.DropoutLayer(network, p=0.5)
network = lasagne.layers.DenseLayer(network, num_units=1)
# optional arguments: nonlinearity, weight initialization, stride, pad, ...

predictions      = lasagne.layers.get_output(network)                                  # symbolic
trainable_params = lasagne.layers.get_all_params(network, trainable=True)              # symbolic
loss             = lasagne.objectives.squared_error(predictions, targets).mean()       # symbolic
adam_updates     = lasagne.updates.adam(loss, trainable_params, learning_rate=0.01)
train_fn = theano.function([inputs, targets], [predictions, loss], updates=adam_updates)

for i in range(10000):                                          # mini-batch training loop
    X,Y = my_minibatch_producer.get()                           # load data
    minibatch_predictions,minibatch_loss = train_fn(X,Y)        # train
```