Theano Introduction

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

Why theano you ask?

- tight integration with NumPy – Use numpy.ndarray in Theano-compiled functions.
- transparent use of a GPU – Perform data-intensive calculations up to 140x faster than with CPU.(float32 only)
- efficient symbolic differentiation – Theano does your derivatives for function with one or many inputs.
- speed and stability optimizations – Get the right answer for log(1+x) even when x is really tiny.
- dynamic C code generation – Evaluate expressions faster.
- extensive unit-testing and self-verification – Detect and diagnose many types of errors.

1. Getting started

1.1 Checking your installation

See if theano imports properly

In [2]:
```
import numpy
import theano.tensor as T
```

1.2 Theano tensors

Theano uses **tensors** to store data. In the examples that follow we shall see some of the common tensor types. These tensors come along with various routines attached with them. Find the list of all the tensors and routines here:

http://deeplearning.net/software/theano/library/tensor/basic.html| (http://deeplearning.net/software/theano/library/tensor/basic.html|)

1.3 Adding two scalars

Reference : http://deeplearning.net/software/theano/tutorial/adding.html (http://deeplearning.net/software/theano/tutorial/adding.html)

In [4]:
```
import numpy
import theano.tensor as T
from theano import function

#Data structure: scalar
x = T.dscalar('x')
y = T.dscalar('y')
z = x + y
f = function([x, y], z+2)

# Execute the function

f(4,9)
```
Out[4]:
```
array(15.0)
```

1.4 Adding two matrices

Reference : http://deeplearning.net/software/theano/tutorial/adding.html (http://deeplearning.net/software/theano/tutorial/adding.html)

In [7]:
```
import numpy as np
import theano.tensor as T
from theano import function

#Data structure: matrix
x = T.dmatrix('x')
y = T.dmatrix('y')
z = x + y
f = function([x, y], z)

# Execute the function
f([[1, 2], [3, 4]], [[10, 20], [30, 40]])
```
Out[7]:
```
array([[ 11.,  22.],
       [ 33.,  44.]])
```

1.4.1 Exercise:

Reference : http://deeplearning.net/software/theano/tutorial/adding.html (http://deeplearning.net/software/theano/tutorial/adding.html)

Write a program that takes a set of 2D points as input and does the following transformation on it:

- x->x+y-1 and y->y-x+1

You can loop over all the points or find a matrix that does this linear transformation and multiply it with the points.

In [6]:
```
import numpy
import theano.tensor as T
from theano import function

# Your code here

x  = T.dmatrix('x')
X = x[0]-x[1]+1
Y = x[0]+x[1]-1
z  = [X,Y]
f = function(x, z)


# END

# Execute the function

f([1,5])
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-6-54922383d93e> in <module>()
      9 Y = x[0]+x[1]-1
     10 z  = [X,Y]
---> 11 f = function(x, z)
     12 
     13 

/root/anaconda2/lib/python2.7/site-packages/theano/compile/function.pyc in func
tion(inputs, outputs, mode, updates, givens, no_default_updates, accept_inplace
, name, rebuild_strict, allow_input_downcast, profile, on_unused_input)
    276         givens = []
    277     if not isinstance(inputs, (list, tuple)):
--> 278         raise Exception("Input variables of a Theano function should be
 "
    279                         "contained in a list, even when there is a sing
le "
    280                         "input.")

Exception: Input variables of a Theano function should be contained in a list,
even when there is a single input.
```

1.5 Shared Variables

It is also possible to make a function with an internal state. For example, let’s say we want to make an accumulator: at the beginning, the state is initialized to zero. Then, on each function call, the state is incremented by the function’s argument.

First let’s define the accumulator function. It adds its argument to the internal state, and returns the old state value.

In [5]:
```
from theano import shared
state = shared(0)
inc = T.iscalar('inc')
accumulator = function([inc], state, updates=[(state, state+inc)])
```

This code introduces a few new concepts. The shared function constructs so-called shared variables. These are hybrid symbolic and non-symbolic variables whose value may be shared between multiple functions. Shared variables can be used in symbolic expressions just like the objects returned by dmatrices(...) but they also have an internal value that defines the value taken by this symbolic variable in all the functions that use it. It is called a shared variable because its value is shared between many functions. The value can be accessed and modified by the .get_value() and .set_value() methods. We will come back to this soon.

The other new thing in this code is the updates parameter of function. updates must be supplied with a list of pairs of the form (shared-variable, new expression). It can also be a dictionary whose keys are shared-variables and values are the new expressions. Either way, it means “whenever this function runs, it will replace the .value of each shared variable with the result of the corresponding expression”. Above, our accumulator replaces the state‘s value with the sum of the state and the increment amount.

Let’s try it out!

In [6]:
```
print(state.get_value())
accumulator(1)
print(state.get_value())
accumulator(300)
print(state.get_value())
```

```
0
1
301
```

It is possible to reset the state. Just use the .set_value() method:

In [7]:
```
state.set_value(-1)
accumulator(3)
print(state.get_value())
```

```
2
```

1.6 Random numbers

The way to think about putting randomness into Theano’s computations is to put random variables in your graph. Theano will allocate a NumPy RandomStream object (a random number generator) for each such variable, and draw from it as necessary. We will call this sort of sequence of random numbers a random stream. Random streams are at their core shared variables, so the observations on shared variables hold here as well. Theanos’s random objects are defined and implemented in RandomStreams and, at a lower level, in RandomStreamsBase.

In [8]:
```
from theano.tensor.shared_randomstreams import RandomStreams
from theano import function
srng = RandomStreams(seed=234)
rv_u = srng.uniform((2,2))
rv_n = srng.normal((2,2))
f = function([], rv_u)
g = function([], rv_n, no_default_updates=True)    #Not updating rv_n.rng
nearly_zeros = function([], rv_u + rv_u - 2 * rv_u)
```

In [9]:
```
f_val0 = f()
f_val1 = f()  #different numbers from f_val0
```

1.7 Gradients

Gradients are very important for all optimization problems. Here we see how to compute simple gradient. For Jacobian and Hessian see the link: http://deeplearning.net/software/theano/tutorial/gradients.html (http://deeplearning.net/software/theano/tutorial/gradients.html)

```
In [10]: import numpy
         import theano
         import theano.tensor as T
         from theano import pp
         x = T.dscalar('x')
         y = x ** 2
         gy = T.grad(y, x)
         pp(gy)  # print out the gradient prior to optimization
         f = theano.function([x], gy)
         f(4)
         numpy.allclose(f(94.2), 188.4)
```

```
Out[10]: True
```

1.8 A simple logistic regression in Theano

Read the code carefully to gain clarity on all the concepts described above.

Reference:http://deeplearning.net/software/theano/tutorial/examples.html#a-real-example-logistic-regression (http://deeplearning.net/software/theano/tutorial/examples.html#a-real-example-logistic-regression)

In [11]:
```
# A basic classifier based on logistic regression

import numpy
import theano
import theano.tensor as T
rng = numpy.random

N = 400                                   # training sample size
feats = 784                               # number of input variables

# generate a dataset: D = (input_values, target_class)
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
training_steps = 10000

# Declare Theano symbolic variables
x = T.dmatrix("x")
y = T.dvector("y")

# initialize the weight vector w randomly
#
# this and the following bias variable b
# are shared so they keep their values
# between training iterations (updates)
w = theano.shared(rng.randn(feats), name="w")

# initialize the bias term
b = theano.shared(0., name="b")

print("Initial model:")
print(w.get_value())
print(b.get_value())

# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))   # Probability that target = 1
prediction = p_1 > 0.5                    # The prediction thresholded
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # Cross-entropy loss function
cost = xent.mean() + 0.01 * (w ** 2).sum()# The cost to minimize
gw, gb = T.grad(cost, [w, b])             # Compute the gradient of the cost
                                          # w.r.t weight vector w and
                                          # bias term b
                                          # (we shall return to this in a
                                          # following section of this tutorial
)

# Compile
train = theano.function(
          inputs=[x,y],
          outputs=[prediction, xent],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))
predict = theano.function(inputs=[x], outputs=prediction)

# Train
for i in range(training_steps):
    pred, err = train(D[0], D[1])

print("Final model:")
print(w.get_value())
print(b.get_value())
print("target values for D:")
print(D[1])
print("prediction on D:")
print(predict(D[0]))
```

```
Initial model:
[  4.85556476e-01  -1.49950362e+00  -1.23397739e+00   3.67761820e-01
   5.88670781e-01  -1.35697131e+00   2.11287504e-01  -2.91961464e-01
   4.07160755e-01   2.04909098e+00  -1.57814122e+00   1.25793266e+00
  -2.78491726e-01   1.35446291e+00  -1.00896760e+00   8.48740851e-01
  -2.26496883e-01   8.76072830e-01   7.20198810e-02   2.88435347e-01
  -2.73699005e+00  -1.95413535e+00   4.31335003e-01   1.09945425e+00
   4.84077724e-01   2.81305874e-01   8.21533164e-01   2.34134904e+00
   2.16959283e+00  -2.31780782e+00  -1.01834544e+00  -4.23139199e-01
   8.08193986e-01  -5.61661011e-01   7.32979953e-01   8.34328126e-01
   8.89500545e-01  -1.44015039e+00   1.23219656e+00   3.49640402e-01
   1.39959078e+00   8.87179569e-01  -1.24479539e+00  -6.51110530e-01
  -2.57699047e+00   2.70417578e+00   6.67113865e-01   1.71528189e+00
   7.64333154e-01  -1.58189644e+00   5.34104994e-01  -1.03387363e+00
   3.17524184e-01   5.57049149e-02   6.12751540e-01   1.07783829e+00
   1.65183972e+00   5.27143773e-01   2.41386107e-01   3.61900863e-02
  -9.57304198e-01  -1.43459788e+00   1.91324368e+00   3.68957719e-01
   5.46621539e-01  -1.69379006e+00  -1.06005000e+00   8.97846378e-01
  -6.56171998e-01   8.22547123e-01   5.14719136e-01  -1.14004144e+00
   7.09968923e-01  -4.78038485e-01   3.37774367e-02  -1.02535789e+00
  -1.82932712e+00  -1.36235610e+00   6.65913810e-01  -7.20895294e-02
  -8.61917688e-01   5.52509958e-01   1.48327575e+00  -1.00962192e+00
   2.90765862e-01   1.09384316e+00   1.25034972e+00  -9.33151338e-01
   2.40576487e-01  -1.17109565e+00  -1.46206395e+00   4.45962576e-01
  -9.32550198e-02  -2.88145520e-01  -4.97787127e-01  -2.07381588e+00
  -4.05821237e-01  -1.23139486e+00   1.73367154e+00  -4.05173240e-01
   5.28224678e-01   2.01053957e-01  -7.96482997e-01  -5.06402730e-01
   6.75340083e-01  -1.32940113e+00   1.51370190e-01   8.85939168e-02
  -1.11177906e+00  -8.79842754e-01   1.35654531e+00  -6.85995477e-01
   5.52951746e-01  -1.75776826e-01   1.39947213e-01   8.53294041e-01
   6.29428346e-01  -4.30703922e-01   1.45800626e+00   5.88616451e-01
   1.13935208e+00   2.68047691e-01   2.33436193e-01   9.85407686e-01
  -2.14707560e-01  -4.07454590e-01  -6.08065365e-01   9.25014703e-01
   1.30566837e+00   1.69463793e+00  -2.53956363e-01   1.13704120e+00
   3.37906181e-01   1.41925680e-01   2.46068972e-01   1.29176422e+00
   1.42768874e+00   9.25440969e-01  -5.46917699e-01  -1.10144156e+00
   7.12855385e-01  -8.18922241e-02   1.72586791e-01  -1.70465941e-01
  -1.34447106e+00  -9.79104088e-01   1.20162488e+00  -4.07913382e-01
   4.28866586e-01  -1.10599017e-01  -3.49350876e-01   6.22869858e-01
  -1.59332243e+00   5.99148333e-01   7.44508506e-01   8.92944534e-01
  -1.94346979e+00   1.73965106e-01   6.09977516e-01   9.05716974e-01
   1.44101704e-01   6.21266869e-01  -2.78821277e-01   1.91437627e+00
   3.44075208e-01   1.40016933e-01  -6.95422587e-01   1.56192549e+00
  -4.46106514e-01  -2.04519574e+00  -1.19624536e+00  -1.18406357e+00
   1.92438434e-01  -1.44221984e+00  -1.47303832e+00  -1.14740738e-01
   2.35812296e-01  -1.39362482e+00  -1.52899746e+00  -6.34726697e-01
   5.29639361e-01   9.43562075e-01   9.34392471e-01   8.10183336e-01
   1.83325389e-01  -1.62286425e+00  -3.76481195e-01   7.41307075e-01
   8.42589036e-01  -1.72414664e+00   5.95337525e-01  -1.92122241e-01
  -3.05576746e+00   1.81791266e+00   2.54862607e-01   1.08051036e+00
   3.30380179e-01   1.15943769e+00  -5.52873563e-01   4.12379497e-01
  -6.17143889e-01  -2.42538761e-01  -1.06515028e+00  -8.53742382e-01
  -1.75255639e+00  -2.06451304e-01  -3.55581258e-02  -1.09546299e+00
   7.59726889e-01   2.85722270e-01   4.68606860e-01   1.70517127e-01
  -2.15622352e-01   2.85429086e-01   1.14083075e+00  -1.57411296e+00
  -8.26794600e-01   1.51785521e+00   1.52187950e-01   4.18340216e-01
  -7.97263736e-01   8.57434436e-01   2.02723874e-01  -2.77853873e-01
   1.94069565e+00   5.07263887e-02  -7.43646928e-01  -1.83816283e+00
  -1.05496954e+00  -6.91062484e-01  -5.30397898e-01  -2.29791426e-01
   9.96811673e-01   1.01045656e+00  -1.99863793e-01   1.02732331e-01
  -1.13404666e+00   1.98341468e+00   7.99457959e-01  -1.56776748e+00
  -3.78494371e-01   2.30701122e+00   9.98723845e-01  -5.77643315e-01
  -1.64439662e-01   8.90943433e-01   3.26412078e-01  -1.22662609e+00
   7.65034128e-01  -1.26497231e-01  -6.01454984e-01  -9.05088024e-01
```

1.8.1 Exercise

Just for the heck of it, try the following:

Q1 : What happens if we modify the cost function?

In [12]:
```
# Your code here
```

Q2 : What happens if we modify the number of training steps?

In [13]:
```
# Your code here
```

There are other functionalities that are native to Theano, but we may not be able to cover all of them here. But you can follow all of them at:

http://deeplearning.net/software/theano/tutorial (http://deeplearning.net/software/theano/tutorial)

It's all good, but.....

Well Theano happens to be a highly customizable and powerful library for machine learning but it can certainly use a lot more abstraction. With this idea in mind an easier to use wrapper was written on top of it to facilitate easier usage without much compromise in functionality. This wrapper happens be Keras. have a look....

Keras Introduction:

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on **top of either TensorFlow or Theano**. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

1. Getting Started¶

Reference:http://keras.io/ (http://keras.io/)

1.1 Checking your Installation

In [14]:
```python
# Check if keras imports successfully
from keras.models import Model
# Import the sequential module from keras
from keras.models import Sequential
# Import the layers you wish to use in your net
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import Input
# Import the optimization algorithms that you wish to use
from keras.optimizers import SGD, Adam, RMSprop
# Import other utilities that help in data formatting etc.
from keras.utils import np_utils
```

```
Using Theano backend.
```

1.2 Generate Data

We shall write a simple logistic regression here to see how Keras works. First generate some random points for data.

In [15]:
```python
import numpy as np
np.random.seed(1337)                            # for reproducibility
N = 400                                        # training sample size
feat = 784                                    # number of input variables
labels=np.random.randint(low=0, high=2, size=(N,1))
x=np.random.randn(N,feat)
x=x.astype('float32')
print(np.shape(labels))
print(np.shape(x))

# convert class vectors to binary class matrices
labels = np_utils.to_categorical(labels, 2)
```

```
(400, 1)
(400, 784)
```

1.3 Building a net

Initiate a sequential model.

In [16]:
```python
model = Sequential()
```

Add the regression neuron

In [17]:
```python
model.add(Dense(2, input_shape=(784,)))
```

Add a non linear activation function

In [18]:
```python
model.add(Activation('sigmoid'))
```

1.4 Compiling the net

In [19]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc
uracy'])
```

1.5 Training the net

In [20]:
```
history = model.fit(x, labels,
                    batch_size=100, nb_epoch=20,
                    verbose=1, validation_data=(x, labels))
```

```
Train on 400 samples, validate on 400 samples
Epoch 1/20
400/400 [==============================] - 0s - loss: 0.8152 - acc: 0.4975 - va
l_loss: 0.7719 - val_acc: 0.5250
Epoch 2/20
400/400 [==============================] - 0s - loss: 0.7605 - acc: 0.5350 - va
l_loss: 0.7286 - val_acc: 0.5475
Epoch 3/20
400/400 [==============================] - 0s - loss: 0.7175 - acc: 0.5525 - va
l_loss: 0.6894 - val_acc: 0.5725
Epoch 4/20
400/400 [==============================] - 0s - loss: 0.6794 - acc: 0.5725 - va
l_loss: 0.6537 - val_acc: 0.5925
Epoch 5/20
400/400 [==============================] - 0s - loss: 0.6435 - acc: 0.6000 - va
l_loss: 0.6217 - val_acc: 0.6175
Epoch 6/20
400/400 [==============================] - 0s - loss: 0.6115 - acc: 0.6275 - va
l_loss: 0.5927 - val_acc: 0.6425
Epoch 7/20
400/400 [==============================] - 0s - loss: 0.5851 - acc: 0.6550 - va
l_loss: 0.5659 - val_acc: 0.6725
Epoch 8/20
400/400 [==============================] - 0s - loss: 0.5573 - acc: 0.6750 - va
l_loss: 0.5423 - val_acc: 0.6925
Epoch 9/20
400/400 [==============================] - 0s - loss: 0.5359 - acc: 0.6925 - va
l_loss: 0.5204 - val_acc: 0.7050
Epoch 10/20
400/400 [==============================] - 0s - loss: 0.5145 - acc: 0.7225 - va
l_loss: 0.5005 - val_acc: 0.7300
Epoch 11/20
400/400 [==============================] - 0s - loss: 0.4953 - acc: 0.7325 - va
l_loss: 0.4822 - val_acc: 0.7575
Epoch 12/20
400/400 [==============================] - 0s - loss: 0.4773 - acc: 0.7675 - va
l_loss: 0.4656 - val_acc: 0.7775
Epoch 13/20
400/400 [==============================] - 0s - loss: 0.4611 - acc: 0.7825 - va
l_loss: 0.4503 - val_acc: 0.8000
Epoch 14/20
400/400 [==============================] - 0s - loss: 0.4461 - acc: 0.8025 - va
l_loss: 0.4360 - val_acc: 0.8050
Epoch 15/20
400/400 [==============================] - 0s - loss: 0.4322 - acc: 0.8100 - va
l_loss: 0.4227 - val_acc: 0.8150
Epoch 16/20
400/400 [==============================] - 0s - loss: 0.4189 - acc: 0.8175 - va
l_loss: 0.4103 - val_acc: 0.8250
Epoch 17/20
400/400 [==============================] - 0s - loss: 0.4069 - acc: 0.8250 - va
l_loss: 0.3985 - val_acc: 0.8350
Epoch 18/20
400/400 [==============================] - 0s - loss: 0.3951 - acc: 0.8375 - va
l_loss: 0.3873 - val_acc: 0.8425
Epoch 19/20
400/400 [==============================] - 0s - loss: 0.3842 - acc: 0.8425 - va
l_loss: 0.3767 - val_acc: 0.8500
Epoch 20/20
400/400 [==============================] - 0s - loss: 0.3737 - acc: 0.8550 - va
l_loss: 0.3666 - val_acc: 0.8575
```

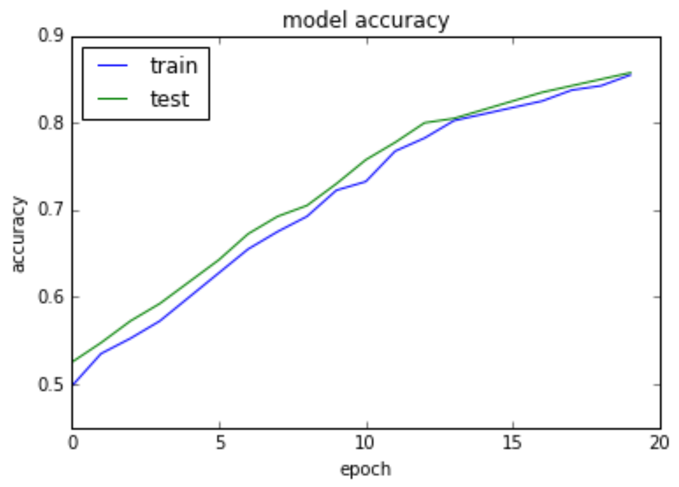
1.6. Plotting accuracy and loss

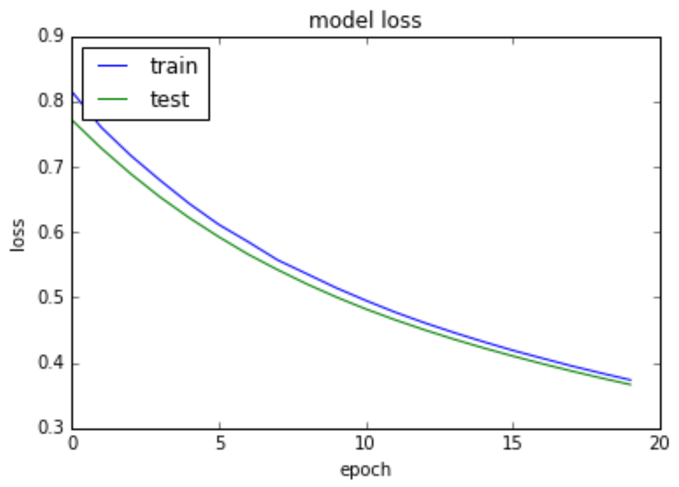
We might be interested in seeing the accuracy or convergence of our model. To do so:

In [21]:
```
import matplotlib.pyplot as plt
%matplotlib inline
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
/root/anaconda2/lib/python2.7/site-packages/matplotlib/font_manager.py:273: Use
rWarning: Matplotlib is building the font cache using fc-list. This may take a
moment.
  warnings.warn('Matplotlib is building the font cache using fc-list. This may
take a moment.')
```

```
['acc', 'loss', 'val_acc', 'val_loss']
```





Well that was quick. Thats how simple Keras is.

2. Exercise

Write a code that reads an image and downsamples it. Hint: You'll need a network with an input layer and a suitable pooling layer.

In [22]:
```
# Your code here
```

Now that the basics are out of the way, lets get serious .....

In [ ]:
```
```