Lecture 3: Theano Programming

Misc Class Items

- Registration & auditing class
- Paper presentation
- Projects:
  - ~10 projects in total
  - ~2 students per project
- AAAI:
  - Hinton's invited talk:
    - Training data size increase
    - Hardware improvement
    - Linear rectifier activation function
    - Pretraining

Review of Last Lecture

- Deep network
- How to learn deep network:
  - Backprop
  - SGD
- Case studies:
  - Logistic Regression
  - MLP
  - CNN

Outline

- Big picture
- Introduction to Theano
- Theano vs. Numpy
- grad() for symbolic expression
- gradient descent as functional programming
- stochastic gradient descent

Big picture

For speech:
- **Kaldi**

For image:
- **Caffe**
- **cuda-convnet (1, 2)**

For general classification:
- **Theano**
- **deeplearning4j**

Deep learning toolkits for three fields

What people say about Theano

- A compiler for mathematical expressions
- Statically typed and functional
- Smartly optimizes symbolic expressions
- Automatically translates them into C++ (or CUDA-C)

What people say about Theano

- A compiler for mathematical expressions

Sounds cool, but we don't know exact what the compiler is doing

- Statically typed and functional

Very general, but its error msgs are hard to understand

- Smartly optimizes symbolic expressions

Easy to use, but sometimes it tries to guess your intention without reporting notification

- Automatically translates them into C++ (or CUDA-C)

Save our labor, but its efficiency is not as good as cuda-convnet

Programming Exercise 1: *Hello World*

- Purpose:
  a. make sure theano + python is installed properly
  b. can import theano & numpy
- Task:
  a. Import theano & numpy
  b. print hello world

Confusing Things about Theano

- Two very similar worlds:
  - Numpy:
    - Matrix operations
    - Vectors & arrays
  - Theano.tensor: needed for symbolic computation and C++ code generation and compilation
    - Matrix operations (applicable to theano.tensor.variables & theano.tensor.constants only)!
    - Theano functions must use theano.tensor operators
    - All numpy arrays/vectors are automatically converted to theano.tensor.constants

Theano Function

```
theano.function(inputs=[x,y], # list of input variables
                outputs=..., # what values to be returned
                updates=..., # “state” values to be modified
                givens=..., # substitutions to the graph
```

Exercise #2: Theano function vs. python

- Purpose:
  - Compare python variable/function vs. theano variable/function
  - Use theano function
- Task:
  - Create a regular python function that implements:
    - f(x,y) = 3x+4y+5
  - Create the theano equivalent function
  - Invoke the two functions with values: <x,y>= <1, 2> & <3, 4>

But what do we care most?

```
from theano import tensor as T

gradients = T.grad(cost, wrt)
```

Automatically compute gradients!!

Examples

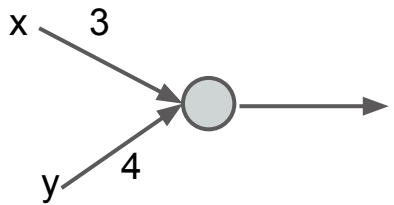
```
import theano.tensor as T
x = T.scalar()

gx = T.grad(x**2, x)

gx2 = T.grad(T.log(x), x)

gx3 = T.grad(1/(x), x)
```

Programming Exercise #3: Gradient

- Purpose:
  a. Demonstrate automatic gradient
  b. Familiarize with the notion of using gradient to find max/min
- Task:
  a. Take the gradient of the function foo in #2 with respect to x, and with respect to y
  b. Take the gradient of the sigmoid of foo with respect to x and with respect to y

Exercise #3 Task b

- What is sigmoid of 3*x+4*y+5?
  - $z = 3*x+4*y+5$
  - $1/(1+e^{-z})$
- Why is this relevant to the topic of this class?



- What's another name for this function?

Why we are excited with automatic gradient?

THE optimization method
(in deep network or large scale learning)

- Stochastic Gradient Descent (SGD)

initialize the model (parameters denoted as ***w***)
do:

- randomly select a sample or a batch
- update $$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla C_{\mathbf{w}}$$

Gradient Descent

- Many ML algorithms aim to find the set of parameters w that minimizes an objective function $C(w)$
- Local minimum can be found by taking a step in the direction of the gradient of $C(w)$ with respect to w
  - $w = w - \alpha \nabla C(w)$
- The step size α is called learning rate
- Online (single example) vs. batch (whole training set) vs. mini batch
- Stochastic gradient descent (SGD)

GD vs. SGD

Gradient descent	Stochastic gradient descent
Converge to local minimum quickly	May dance around local minimum
All the samples should be in memory	Load one sample (or a batch) at a time
Not scalable	Scalable but take account into disk IO
More likely to get into local minimum	Higher chance to jump out local minimum due to randomness

Gradient descent as functional programming

```
W = theano.shared(value=np.zeros((n_in, n_out)),  name='W', borrow=True)
b = theano.shared(value=np.zeros((n_out,)), name = 'b')

g_W = T.grad(cost, W)
g_b =  T.grad(cost, b)

updates_FP = [(self.W, W - learning_rate * g_W),
                    (self.b, b - learning_rate * g_b)]

train_model = theano.function(inputs=[x,y], outputs=cost_func, updates=updates_FP )
```

train_model can be called easily during SGD training

```
for every epoch
        for every batch (xi,yi)
                train_model(xi,yi)
```

Exercise #4: Logistic Regression Through Gradient Descent

- Purpose:
  - Familiarize with gradient descent algorithm
- Task: given a set of training instances $<x_j, y_i>$ from MNIST data set, implement a multi nomial logistic regression model using a mini-batch gradient descent that stops after 1 epoch:
  - soft max function: $e^{w\_i * x}/sum(e^{w\_k * x})$
- Use the following cost function:
  - negative log likelihood/cross entry/log loss:
    - $-log(y_i)$

Let’s put things together

```
class lr_model(object):
    def __init__(self, x, n_in, n_out):
      ...
    def get_cost(self, y):
      ...
    def get_estimation(self):
      ...
    def get_prediction(self):
      ...
```

```
def do_sgd:
      # load mnist data
      # define theano functions
      # training
      ……
```

```
def __init__(self, x, n_in, n_out):
        self.W = theano.shared(value=numpy.zeros((n_in, n_out),
                                                 dtype='float64'),
                               name='W', borrow=True)
        self.b = theano.shared(value=numpy.zeros((n_out,),
                                                 dtype='float64'),
                               name='b', borrow=True)


        # compute vector of class-membership probabilities in symbolic
    form
        self.y_esti = T.nnet.softmax(T.dot(x, self.W) + self.b)
```

def do_sgd:

```
dataset = '../data/mnist.pkl.gz'
f = gzip.open(dataset, 'rb')
train_set, valid_set, test_set = cPickle.load(f)
f.close()

train_set_x, train_set_y = train_set
shared_train_x = theano.shared(numpy.asarray(train_set_x,
                                             dtype='float64'),
                               borrow=True)
... ...
```

Debugging Theano

- Cannot step through Theanao functions
- There are some built in facilities for debugging:
  - theano.config.exception_verbosity='high‘
  - theano.config.compute_test_value = 'warn‘
  - Etc.
- Debug by simplifying the functions to the basic operations

Evaluate a Theano expression

```
import theano.tensor as T
Import numpy as np


vx = T.vector()
fx = vx**2
fx.eval({vx:np.array([2,3,0])})
```

Conclusion

- Theano is a powerful tool
  - but sometimes difficult to debug

- SGD + NN training is easy to implement
  - But there are some tricks to improve the performance

A few errors from my experiments

- Theano
  - Theano can only use simple indexing, and (for newest version) integer indexing
  - Numpy can use Boolean vector for indexing, but theano cannot!
  - Errors may or may not be reported
- numpy

```
import numpy as np
a = np.array([1,2])
b = np.ones((3,1))
print a + b
```

Guess what is the result?