Lab 4

- Tuan Nguyen
- Deep learning Spring 2017
- Dr. Martin Hagan

Debugging one_layer_theano.py

In [1]:
```
import numpy as np
import theano
import theano.tensor as T
```

In [2]:
```
rng = np.random
```

In [3]:
```
input = np.array([[1],[2],[3],[4]])
target = np.array([[3],[5],[7],[9]])
```

--> Input and Target values (very small dataset with 4 data points)

In [4]:
```
p = T.dmatrix("p")
t = T.dmatrix("t")
```

--> Just like placeholders in TensorFlow, p and t are symbolic matrixes (that will be fed values later)

In [5]:
```
# initialize the weight
w = theano.shared(rng.randn(1), name="w")
# initialize the bias
b = theano.shared(0., name="b")
```

--> W and b are weight and bias, just like Variable in TensorFlow

In [6]:
```
iterations = 1000
a = w*p + b
```

--> a linear layer (perceptron) with single neuron

In [7]:
```
e = t - a
e2 = T.sqr(e)
perf = T.sum(e2)
```

--> compute performance, which is sum square error between a (actual output) and t (target)

In [8]:
```
gw, gb = T.grad(perf, [w, b])
```

--> compute gradient of the performance, respect to weight and bias

In [9]:
```
train = theano.function(
          inputs=[p,t],
          outputs=[a, perf],
          updates=((w, w - 0.01 * gw), (b, b - 0.01 * gb)))
predict = theano.function(inputs=[p], outputs=a)
perform = theano.function(inputs=[p,t], outputs=perf)

# Train,
for i in range(iterations):
    pred, err = train(input, target)
```

--> declare a train function then run train function for 1000 times

In [10]:
```
print("Final model:")
print(w.get_value())
print(b.get_value())
```

```
Final model:
[ 1.99999918]
1.00000239624
```

The network tried to approximate linear equation y = 2 * x + 1. And the weight (1.99) and bias (1.00) are very close to the correct ones.