**Improving the way neural networks learn**

The techniques that we’ll develop in this chapter:

1. A better choice of cost function, known as cross-entropy cost function.
2. Four regularization methods: L1 and L2 regularization, dropout, artificial expansion of the training data.
3. A better method for initializing the weights in the network.
4. A set of heuristics to help choose good hyper-parameters for the network.

**3.1. The cross-entropy cost function**

import matplotlib.pylab as plt

import numpy as np

class Sigmoid(object):

def activate(self, z):

return 1. / (1. + np.exp(-z))

def diff(self, z):

return self.activate(z) \* (1. - self.activate(z))

class QuadraticCost():

def cost(self, y, out):

return 0.5 \* ((y - out) \*\* 2.)

def diff(self, y, out):

return (out - y)

class CrossEntropyCost(object):

def cost(self, y, out):

return -(y \* np.log(out) + (1. - y) \* np.log(1. - out))

def diff(self, y, out):

return (y - out) / (out \* (out - 1.))

def singleNeuronModel(weight, bias, x=1.0, y=0.0, costFunction=QuadraticCost(),eta=0.15, activationFunction=Sigmoid(),epoch=300):

allCosts = []

for i in range(epoch):

z = x \* weight+bias

out = activationFunction.activate(z)

weight -= eta \* costFunction.diff(y, out) \* activationFunction.diff(z) \* out

bias -= eta \* costFunction.diff(y, out) \* activationFunction.diff(z)

allCosts.append(costFunction.cost(y, out))

print "weight: {}, bias: {}, output: {}".format (weight, bias, allCosts[-1])

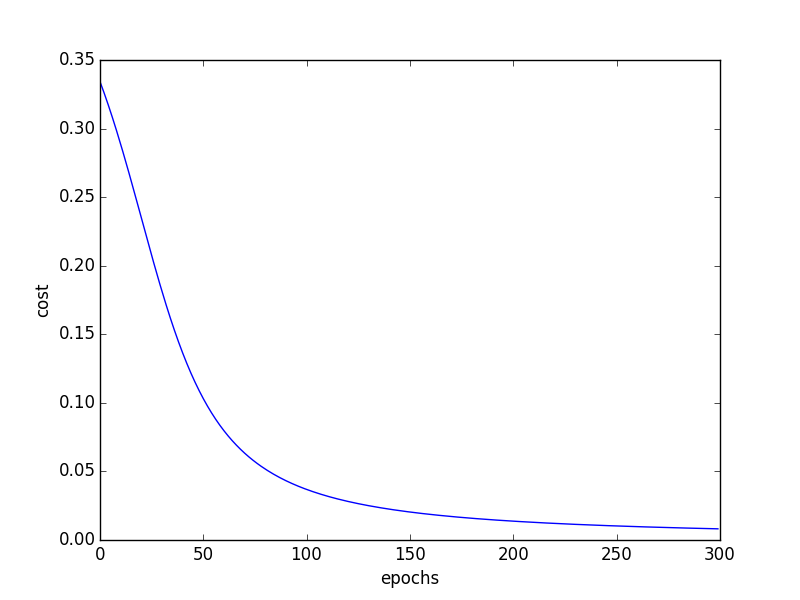
plt.plot(range(epoch), allCosts)

plt.xlabel('epochs')

plt.ylabel('cost')

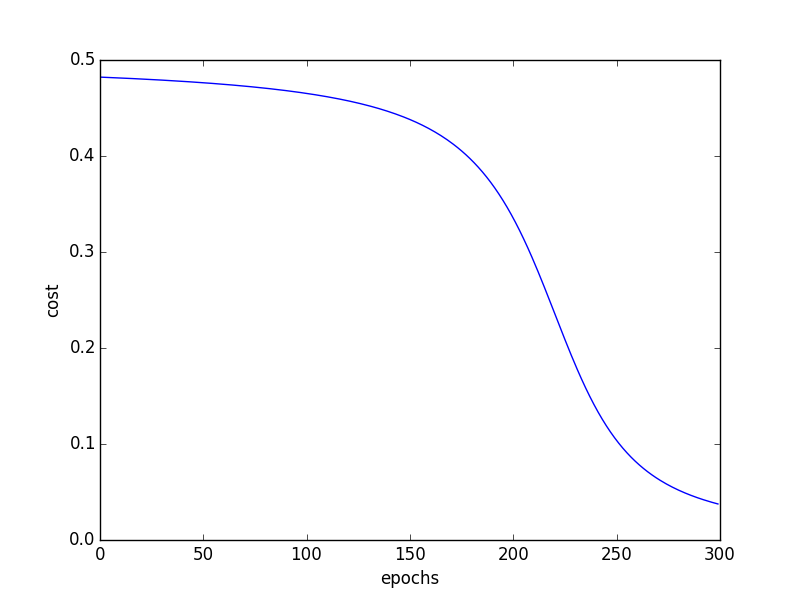
plt.show()

singleNeuronModel(weight=0.6, bias=0.9)



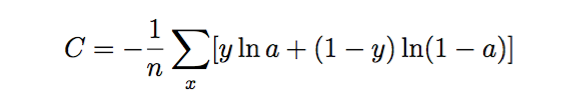
weight: -0.424803180931, bias: -1.49947387899, output: 0.00814763047625

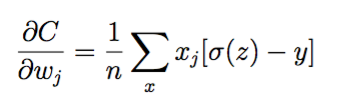
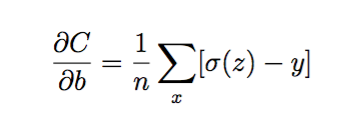
singleNeuronModel(weight=2.0, bias=2.0)



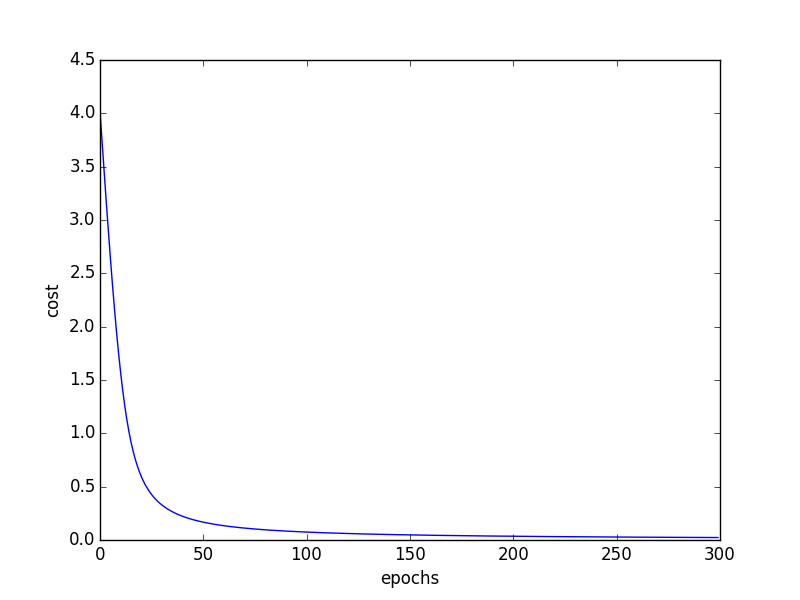
weight: -0.075825438188, bias: -0.909052233929, output: 0.0375357988199

**3.2. Introducing the cross-entropy cost function**

****

****

singleNeuronModel(weight=2.0, bias=2.0, costFunction=CrossEntropyCost())



As you can see with cross-entropy cost function, the neuron learned quickly, although the bias and weight is equal to 2.

singleNeuronModel(weight=0.6, bias=0.9, costFunction=CrossEntropyCost())

