CSC321
Tutorial 2: Implementing a logistic neuron in Python

Shikhar Sharma

January 20, 2015

Loss function L for the logistic neuron is given by:

$$L = \frac{1}{2}\sum_n (y^{(n)} - t^{(n)})^2 \tag{1}$$

where $y^{(n)}$ **respresents the y value of the n^{th} training example**. As taught in the lecture notes, for a single training example:

$$z = \sum_i x_i w_i + b \tag{2}$$

$$y = \frac{1}{1+e^{-z}} \tag{3}$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i \qquad \frac{dy}{dz} = y(1-y) \tag{4}$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i\, y(1-y) \tag{5}$$

$$\frac{\partial L}{\partial w_i} = \sum_n \frac{\partial y^{(n)}}{\partial w_i}\frac{dL}{dy^{(n)}} = \sum_n x_i^{(n)} y^{(n)}(1-y^{(n)})(y^{(n)} - t^{(n)}) \tag{6}$$

$$\Delta w_i = -\epsilon \frac{\partial L}{\partial w_i} = -\epsilon \sum_n x_i^{(n)} y^{(n)}(1-y^{(n)})(y^{(n)} - t^{(n)}) \tag{7}$$

$$\Delta b = -\epsilon \sum_n y^{(n)}(1-y^{(n)})(y^{(n)} - t^{(n)}) \tag{8}$$

Instead of looping over all the training examples, it is much more efficient to compute these values in one go using vectors. In the vector notation for the code example shown:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \tag{9}$$

$$\mathbf{X} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ \vdots & \vdots \\ x_0^{(1000)} & x_1^{(1000)} \end{bmatrix} \tag{10}$$

$$\mathbf{z} = \mathbf{X}\cdot\mathbf{w} + \mathbf{b} = \begin{bmatrix} x_0^{(1)} w_0 + x_1^{(1)} w_1 + b \\ \vdots \\ x_0^{(1000)} w_0 + x_1^{(1000)} w_1 + b \end{bmatrix} = \begin{bmatrix} \sum_i x_i^{(1)} w_i + b \\ \vdots \\ \sum_i x_i^{(1000)} w_i + b \end{bmatrix} = \begin{bmatrix} z^{(1)} \\ \vdots \\ z^{(1000)} \end{bmatrix} \tag{11}$$

which is written as:

```
z = np.dot(x, w)+b
```

$$\mathbf{y} = \begin{bmatrix} \frac{1}{1+e^{-z^{(1)}}} \\ \vdots \\ \frac{1}{1+e^{-z^{(1000)}}} \end{bmatrix} \tag{12}$$

which is written as:

```
y = 1/(1+np.exp(-z))
```

$$\mathbf{L} = \frac{1}{2}(\mathbf{y}-\mathbf{t})^2 = \frac{1}{2}\begin{bmatrix} y^{(1)} - t^{(1)} \\ \vdots \\ y^{(1000)} - t^{(1000)} \end{bmatrix}^2 \tag{13}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{y}} = \mathbf{y} - \mathbf{t} = \begin{bmatrix} y^{(1)} - t^{(1)} \\ \vdots \\ y^{(1000)} - t^{(1000)} \end{bmatrix} \tag{14}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{z}} = \frac{d\mathbf{y}}{d\mathbf{z}}\frac{\partial \mathbf{L}}{\partial \mathbf{y}} = \left(\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(1000)} \end{bmatrix} \bullet \begin{bmatrix} 1 - y^{(1)} \\ \vdots \\ 1 - y^{(1000)} \end{bmatrix} \right) \bullet \begin{bmatrix} y^{(1)} - t^{(1)} \\ \vdots \\ y^{(1000)} - t^{(1000)} \end{bmatrix} \tag{15}$$

which is written as:

```
dLbydy = y-t
dLbydz = dLbydy * y * (1-y)
```

We know that

$$\Delta w_i = -\epsilon \frac{\partial L}{\partial w_i} = -\epsilon \sum_n x_i^{(n)} y^{(n)} (1 - y^{(n)})(y^{(n)} - t^{(n)}) \tag{16}$$

$$\Delta \mathbf{w} = -\epsilon \frac{\partial \mathbf{L}}{\partial \mathbf{w}} = -\epsilon \begin{bmatrix} x_0^{(1)} & \dots & x_0^{(1000)} \\ x_1^{(1)} & \dots & x_1^{(1000)} \end{bmatrix} \cdot \begin{bmatrix} y^{(1)}(1 - y^{(1)})(y^{(1)} - t^{(1)}) \\ \vdots \\ y^{(1000)}(1 - y^{(1000)})(y^{(1000)} - t^{(1000)}) \end{bmatrix} = \begin{bmatrix} \Delta w_0 \\ \Delta w_1 \end{bmatrix} \tag{17}$$

$$\Delta b = -\epsilon \sum_n y^{(n)}(1 - y^{(n)})(y^{(n)} - t^{(n)}) = -\epsilon \, sum(\frac{\partial \mathbf{L}}{\partial \mathbf{z}}) \tag{18}$$

which is written as:

```
dLbydw = np.dot(x.T, dLbydz)
dLbydb = np.sum(dLbydz)

dw = -epsilon * dLbydw
db = -epsilon * dLbydb

w = w + dw
b = b + db
```

The complete code for the tutorial is:

```
import numpy as np
import matplotlib.pyplot as plt
import time

def make_data():
        data0 = np.tile([1,2],(500,1))+0.5*np.random.randn(500,2)
        data1 = np.tile([3,1],(500,1))+0.5*np.random.randn(500,2)
        data = np.concatenate((data0,data1))
        return data

def make_labels():
        labels = np.concatenate((np.zeros((500,1)),np.ones((500,1))))
        return labels

def plot_data(data, labels):
        plt.scatter(data[:,0], data[:,1], s=10, c=labels, edgecolors='none')
        plt.show()

def show_neuron(data, labels, w, b):
        plt.clf()
        plt.scatter(data[:,0], data[:,1], s=10, c=labels, edgecolors='none')

        xmin = np.min(data[:,0])
        xmax = np.max(data[:,0])

        x = np.arange(xmin,xmax+0.01,0.01)
        y = -b/w[1] - (w[0]/w[1])*x

        plt.plot(x,y,'g')
        plt.xlim((-1,5))
        plt.ylim((-1.5,4.5))
        plt.draw()
        time.sleep(0.01)

def learn_neuron(data, labels):
        w = 0.01*np.random.randn(2,1)
        b = 0
        epsilon = 0.0001

        x = data
        t = labels

        for i in xrange(1000):
                show_neuron(data,labels,w,b)
                plt.pause(0.1)
                z = np.dot(x, w)+b
                y = 1/(1+np.exp(-z))

                prediction = y > 0.5
                acc = np.mean(prediction.astype(int)==t)

                L = 0.5*np.sum(np.square(y-t))
                print 'w[0] = ', w[0], ' w[1] = ', w[1], ' b = ', b, ' L = ', np.sum(L), \
                 ' acc = ', acc

                # Compute dw, db
                dLbydy = y-t
                dLbydz = dLbydy * y * (1-y)
                dLbydw = np.dot(x.T, dLbydz)
                dLbydb = np.sum(dLbydz)
                dw = -epsilon * dLbydw
                db = -epsilon * dLbydb

                w = w + dw
                b = b + db
        plt.ioff()
        plt.show()


plt.ion()
data = make_data()
labels = make_labels()
plot_data(data,labels)
learn_neuron(data, labels)
```