Simple Neural Network

This is a draft document when I'm learning neural network. It will be polished in several days.

Before you reading this article, you should have some knowledge of gradient descent, it's highly recommended to checkout out this paper "Gradient Descent Example Code" before walking through the following content.

1 what is neural network?

2 Talk is cheap, show me the code.

```
import numpy as np

def nonlin(x,deriv=False):
    if(deriv==True):
        return x*(1-x)

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

def test(x):
    l1 = nonlin(np.dot(x,syn0))
    return nonlin(np.dot(l1,syn1))

X = np.array([[0,0,0], [0,0,1], [0,1,0], [0,1,1],
              [1,0,0], [1,0,1], [1,1,0], [1,1,1]])
y = np.array([[1,0,0,0,0,0,0,1]]).T

np.random.seed(1)

syn0 = 2*np.random.random((3,11)) - 1
syn1 = 2*np.random.random((11,1)) - 1

for j in xrange(60000):

    # Feed forward through layers 0, 1, and 2
    l0 = X
    l1 = nonlin(np.dot(l0,syn0))
    l2 = nonlin(np.dot(l1,syn1))

    # how much did we miss the target value?
    l2_error = y - l2

    # in what direction is the target value?
    # were we really sure? if so, don't change too much.
    l2_delta = l2_error*nonlin(l2,deriv=True)
```

```
    # how much did each l1 value contribute to the l2 error (according to the weights)?
    l1_error = l2_delta.dot(syn1.T)
    # l1_error = l2_error*nonlin(l2,deriv=True).dot(syn1.T)

    # in what direction is the target l1?
    # were we really sure? if so, don't change too much.
    l1_delta = l1_error * nonlin(l1,deriv=True)
    # l1_delta = l2_error*nonlin(l2,deriv=True).dot(syn1.T)*nonlin(l1,deriv=True)

    syn1 += l1.T.dot(l2_delta)
    syn0 += l0.T.dot(l1_delta)

print "Error:" + str(np.mean(np.abs(l2_error)))
for x in X:
    print test(x)

print test([-1,-1,-1])
print test([2,2,2])
print test([3,3,3])
```

3 Why update weight matrix syn0 and syn1 in that way?

Here we have m samples S in n dimensions space and corresponding labels Y.

$$samples: S_{m*n} = \begin{bmatrix} S_{11} & \dots & S_{1n} \\ \dots & \dots & \dots \\ S_{m1} & \dots & S_{mn} \end{bmatrix}; targets: Y_{m*1} = \begin{bmatrix} Y_1 \\ \dots \\ Y_m \end{bmatrix}$$

First layer of weights, synapse 0, connecting input layer to hidden layer:

$$syn0_{n*p} = \begin{bmatrix} syn0_{11} & \dots & syn0_{1p} \\ \dots & \dots & \dots \\ syn0_{n1} & \dots & syn0_{np} \end{bmatrix}$$

We got temporary hidden layer before squashing:

$$Hr_{m*p} = \begin{bmatrix} Hr_{11} & \dots & Hr_{1p} \\ \dots & \dots & \dots \\ Hr_{m1} & \dots & Hr_{mp} \end{bmatrix}, Hr = S \cdot syn0, Hr_{xy} = \sum_{i=1}^{i=n} S_{xi} * syn0_{iy},$$

This is the hidden layer, will be projected to the output layer:

$$H_{m*p} = \begin{bmatrix} H_{11} & \dots & H_{1p} \\ \dots & \dots & \dots \\ H_{m1} & \dots & H_{mp} \end{bmatrix}, H_{xy} = \frac{1}{1 + e^{-Hr_{xy}}}$$

$$syn1_{p*1} = \begin{bmatrix} syn1_1 \\ ..... \\ syn1_p \end{bmatrix}$$

$$Or_{m*1} = \begin{bmatrix} Or_1 \\ .... \\ Or_m \end{bmatrix}, Or = H \cdot syn2, Or_x = \sum_{i=1}^{i=p} H_{xi} * syn1_i$$

This is the output layer, or the prediction.

$$O_{m*1} = \begin{bmatrix} O_1 \\ ... \\ O_m \end{bmatrix}, Ox = \frac{1}{1+e^{-Or_x}}$$

Finally, the cost or loss function can be expressed like this:

$$L = \frac{1}{2} \sum_{i=1}^{i=m} (O_i - Y_i)^2 \tag{3.0.1}$$

Firstly, we need to update the second weight matrix which connecting the hidden and output layer.

We take sample S_i as example, the prediction is H_i, we can calculate the derivative of weight matrix syn1 by applying the chain rule we know that:

$$\frac{\partial L}{\partial syn1_x} = \frac{\partial L}{\partial O_i} * \frac{\partial O_i}{\partial Or_i} * \frac{\partial Or_i}{\partial syn1_x} \tag{3.0.2}$$

$$grad_i = \begin{bmatrix} \frac{\partial L}{\partial syn1_1} \\ ..... \\ \frac{\partial L}{\partial syn1_p} \end{bmatrix} \tag{3.0.3}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial O_i} * \frac{\partial O_i}{\partial Or_i} * \frac{\partial Or_i}{\partial syn1_1} \\ ..... \\ \frac{\partial L}{\partial O_i} * \frac{\partial O_i}{\partial Or_i} * \frac{\partial Or_i}{\partial syn1_p} \end{bmatrix} \tag{3.0.4}$$

$$= \begin{bmatrix} (O_i - Y_i) * O_i * (1 - O_i) * H_{i1} \\ ..... \\ (O_i - Y_i) * O_i * (1 - O_i) * H_{ip}] \end{bmatrix} \tag{3.0.5}$$

$$= (O_i - Y_i) * O_i * (1 - O_i) * H_i^T. \tag{3.0.6}$$

H_i^T is the ith row of hidden layer H^T

then we update weight matrix $syn1$ through the gradient,

$$syn1' = syn1 + (-1) * grad_i * alpha$$

,

alpha is the step size, we can update $syn1$ in a batch way.

$$syn1' = syn1 + (-1) * alpha * \sum_{i=1}^{m} grad_i \tag{3.0.7}$$

$$= syn1 + (-1) * alpha * \sum_{i=1}^{m} (O_i - Y_i) * O_i * (1 - O_i) * H_i^T \tag{3.0.8}$$

$$= syn1 + (-1) * alpha * H^T \cdot (O - Y) * O * (1 - O) \tag{3.0.9}$$

like above, we now update syn0 with sample S_i:

$$\frac{\partial L}{\partial syn0_{xy}} = \frac{\partial L}{\partial O_i} * \frac{\partial O_i}{\partial Or_i} * \frac{\partial Or_i}{\partial H_{iy}} * \frac{\partial H_{iy}}{\partial Hr_{iy}} * \frac{\partial Hr_{iy}}{\partial syn0_{xy}}$$

since $syn0_{11}$ only exists in Hr_{x1}, here is the sample id, $1 \leq x \leq m$.

for sample S_i,

$$Hr_{iy} = S_{ix} \cdot syn0_{xy}$$

,

$$H_{iy} = \frac{1}{1 + e^{-Hr_{iy}}}$$

,

$$Or_i = \sum_{j=1}^{p} H_{ij} * syn1_j$$

so finaly

$$\frac{\partial L}{\partial syn0_{xy}} = (O_i - Y_i) * O_i * (1 - O_i) * syn1_y * H_{iy} * (1 - H_{iy}) * S_{ix}$$

$$\begin{bmatrix} \frac{\partial L}{\partial syn0_{11}} & \cdots & \frac{\partial L}{\partial syn0_{1p}} \\ \cdots & \cdots & \cdots \\ \frac{\partial L}{\partial syn0_{n1}} & \cdots & \frac{\partial L}{\partial syn0_{np}} \end{bmatrix} \tag{3.0.10}$$

$$= (O_i - Y_i) * O_i * (1 - O_i) * \left(\begin{bmatrix} S_{i1} \\ \dots \\ S_{in} \end{bmatrix} \cdot \begin{bmatrix} syn1_1 * H_{i1} * (1 - H_{i1}) & \dots & syn1_p * H_{ip} * (1 - H_{ip}) \end{bmatrix} \right) \tag{3.0.11}$$

$$= (O_i - Y_i) * O_i * (1 - O_i) * (S_i^T \cdot (syn1^T * H_i * (1 - H_i)) \tag{3.0.12}$$

$$= S_i^T \cdot ((O_i - Y_i) * O_i * (1 - O_i) * syn1^T * H_i * (1 - H_i)) \tag{3.0.13}$$

do batch update, so

$$syn0' = syn0 + (-1) * alpha * \sum_{i=1}^{m} S_i^T \cdot ((O_i - Y_i) * O_i * (1 - O_i) * syn1^T * H_i * (1 - H_i)) \tag{3.0.14}$$

$$= syn0 + (-1) * alpha * S^T \cdot (\begin{bmatrix} (O_1 - Y_1) * O_1 * (1 - O_1) * syn1^T * H_1 * (1 - H_1) \\ \dots\dots\dots\dots\dots\dots \\ (O_m - Y_m) * O_m * (1 - O_m) * syn1^T * H_m * (1 - H_m) \end{bmatrix}) \tag{3.0.15}$$

$$= syn0 + (-1) * alpha * S^T \cdot (\begin{bmatrix} (O_1 - Y_1) * O_1 * (1 - O_1) \\ \dots\dots\dots \\ (O_m - Y_m) * O_m * (1 - O_m) \end{bmatrix} * \begin{bmatrix} syn1^T \\ \dots\dots \\ syn1^T \end{bmatrix} * \begin{bmatrix} H_1 * (1 - H_1) \\ \dots\dots \\ H_m * (1 - H_m) \end{bmatrix}) \tag{3.0.16}$$

$$= syn0 + (-1) * alpha * S^T \cdot (((O - Y) * O * (1 - O)) \cdot syn1^T * (H * (1 - H)) \tag{3.0.17}$$

You may be confused about equation 3.0.17, here we will explain in detail:

$$\begin{bmatrix} (O_1 - Y_1) * O_1 * (1 - O_1) \\ \dots\dots\dots \\ (O_m - Y_m) * O_m * (1 - O_m) \end{bmatrix} * \begin{bmatrix} syn1^T \\ \dots\dots \\ syn1^T \end{bmatrix}$$
$$= \begin{bmatrix} (O_1 - Y_1) * O_1 * (1 - O_1) * syn1_1^T & \dots & (O_1 - Y_1) * O_1 * (1 - O_1) * syn1_m^T \\ \dots & \dots & \dots \\ (O_m - Y_m) * O_m * (1 - O_m) * syn1_1^T & \dots & (O_m - Y_m) * O_m * (1 - O_m) * syn1_m^T \end{bmatrix} \tag{3.0.18}$$
$$= ((O - Y) * O * (1 - O)) \cdot syn1^T$$

3.1 Future Work