The Maths behind Neural Networks

Alex Punnen
© All Rights Reserved

Contents

The Maths behind Neural Networks

Alex Punnen
© All Rights Reserved

Contents

Chapter 6

A Simple NeuralNet with Back Propagation

With the derivative of the Cost function dervied from the last chapter, we can code the network

We will use matrices to represent input and weight matrices.

```
x = np.array(
    [
        [0,0,1],
        [0,1,1],
        [1,0,1],
        [1,1,1]
    ])
```

This is a 4*3 matrix. Note that each row is an input. lets take all this 4 as 'training set'

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

Note you can change the output and try to train the Neural network

This is a 4*1 matrix that represent the expected output. That is for input [0,0,1] the output is [0] and for [0,1,1] the output is [1] etc.

A neural network is implemented as a set of matrices representing the weights of the network.

Let's create a two layered network. Before that please not the formula for the neural network

So basically the output at layer l is the dot product of the weight matrix of layer l and input of the previous layer.

Now let's see how the matrix dot product works based on the shape of matrices.

```
[m*n].[n*x] = [m*x]
[m*x].[x*y] = [m*y]
```

We take the $[m * n]$ as the input matrix this is a $[4 * 3]$ matrix.

Similarly the output y is a $[4 * 1]$ matrix; so we have $[m * y] = [4 * 1]$

So we have

```
m=4
n=3
x=?
y=1
```

Lets then create our two weight matrices of the above shapes, that represent the two layers of the neural network.

```
w0 = x
w1 = np.random.random((3,4))
w2 = np.random.random((4,1))
```

We can have an array of the weights to loop through, but for the time being let's hard-code these. Note that 'np' stands for the popular numpy array library in Python.

We also need to code in our non linearity.We will use the Sigmoid function here.

```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

```
# derivative of the sigmoid
def derv_sigmoid(x):
   return sigmoid(x)*(1-sigmoid(x))
```

With this we can have the output of first, second and third layer, using our equation of neural network forward propagation.

```
a0 = x
a1 = sigmoid(np.dot(a0,w1))

a2 = sigmoid(np.dot(a1,w2))
```

a2 is the calculated output from randomly initialized weights. So lets calculate the error by subtracting this from the expected value and taking the MSE.

$$C = \frac{1}{2}\|y - a^l\|^2$$

```
c0 = ((y-a2)**2)/2
```

Now we need to use the back-propagation algorithm to calculate how each weight has influenced the error and reduce it proportionally.

We use this to update weights in all the layers and do forward pass again, re-calculate the error and loss, then re-calculate the error gradient $\frac{\partial C}{\partial w}$ and repeat

$$w^2 = w^2 - (\frac{\partial C}{\partial w^2}) * learningRate$$

$$w^1 = w^1 - (\frac{\partial C}{\partial w^1}) * learningRate$$

Let's update the weights as per the formula (3) and (5) from last chapter

$$\frac{\mathbf{C}}{\mathbf{w^1}} = \ '(\mathbf{z^1}) * (\mathbf{a^0})^{\mathbf{T}} * \ ^{\mathbf{2}} * \mathbf{w^2}.\ '(\mathbf{z^2}) \quad \rightarrow \mathbb{E}\mathbf{q}\ (\mathbf{5})$$

$$\delta^2 = (a^2 - y)$$

$$\frac{\mathbf{C}}{\mathbf{w^2}} = \ ^{\mathbf{2}} * \ '(\mathbf{z^2}) * (\mathbf{a^1})^{\mathbf{T}} \quad \rightarrow \mathbb{E}\mathbf{q}\ (\mathbf{3})$$

A Two layered Neural Network in Python

Below is a two layered Network; I have used the code from http://iamtrask.github.io/2015/07/12/basic-python-network/ as the basis. With minor changes to fit into how we derived the equations.

```
import numpy as np
# seed random numbers to make calculation deterministic
np.random.seed(1)

# pretty print numpy array
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})

# let us code our sigmoid funciton
def sigmoid(x):
    return 1/(1+np.exp(-x))

# let us add a method that takes the derivative of x as well
def derv_sigmoid(x):
   return sigmoid(x)*(1-sigmoid(x))

#---------------------------------------------------------------

# Two layered NW. Using from (1) and the equations we derived as explanaionns
# (1) http://iamtrask.github.io/2015/07/12/basic-python-network/
#---------------------------------------------------------------

# set learning rate as 1 for this toy example
learningRate = 1

# input x, also used as the training set here
x = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])

# desired output for each of the training set above
y = np.array([[0,1,1,0]]).T

# Explanaiton - as long as input has two ones, but not three, ouput is One
"""
Input [0,0,1]  Output = 0
Input [0,1,1]  Output = 1
Input [1,0,1]  Output = 1
Input [1,1,1]  Output = 0
"""

# Randomly initalised weights
weight1 =  np.random.random((3,4))
```

```
weight2 =  np.random.random((4,1))

# Activation to layer 0 is taken as input x
a0 = x

iterations = 1000
for iter in range(0,iterations):

  # Forward pass - Straight Forward
  z1= np.dot(x,weight1)
  a1 = sigmoid(z1)
  z2= np.dot(a1,weight2)
  a2 = sigmoid(z2)
  if iter == 0:
    print("Intial Ouput \n",a2)

  # Backward Pass - Backpropagation
  delta2  = (a2-y)
  #---------------------------------------------------------------
  # Calcluating change of Cost/Loss wrto weight of 2nd/last layer
  # Eq (A) ---> dC_dw2 = delta2*derv_sigmoid(z2)*a1.T
  #---------------------------------------------------------------

  dC_dw2_1  = delta2*derv_sigmoid(z2)
  dC_dw2  = a1.T.dot(dC_dw2_1)

  #---------------------------------------------------------------
  # Calcluating change of Cost/Loss wrto weight of 2nd/last layer
  # Eq (B)---> dC_dw1 = derv_sigmoid(z1)*delta2*derv_sigmoid(z2)*weight2*a0.T
  # dC_dw1 = derv_sigmoid(z1)*dC_dw2*weight2_1*a0.T
  #---------------------------------------------------------------

  dC_dw1 =  np.multiply(dC_dw2_1,weight2.T) * derv_sigmoid(z1)
  # todo - the weight2.T is the only thing not in equation here
  dC_dw1 = a0.T.dot(dC_dw1)

  #---------------------------------------------------------------
  #Gradinent descent
  #---------------------------------------------------------------

  weight2 = weight2 - learningRate*(dC_dw2)
  weight1 = weight1 - learningRate*(dC_dw1)


print("New ouput",a2)
```

```
#-------------------------------------------------------------
# Training is done, weight2 and weight2 are primed for output y
#-------------------------------------------------------------

# Lets test out, two ones in input and one zero, ouput should be One
x = np.array([[1,0,1]])
z1= np.dot(x,weight1)
a1 = sigmoid(z1)
z2= np.dot(a1,weight2)
a2 = sigmoid(z2)
print("Ouput after Training is \n",a2)
```

Output

```
Intial Ouput
 [[ 0.758]
 [ 0.771]
 [ 0.791]
 [ 0.801]]
New ouput [[ 0.028]
 [ 0.925]
 [ 0.925]
 [ 0.090]]
Ouput after Training is
 [[ 0.925]]
```

We have trained the NW for getting the output similar to y; that is [0,1,0,1]

The code in Colab

Next: Back Propagation Pass 3 (Matrix Calculus)