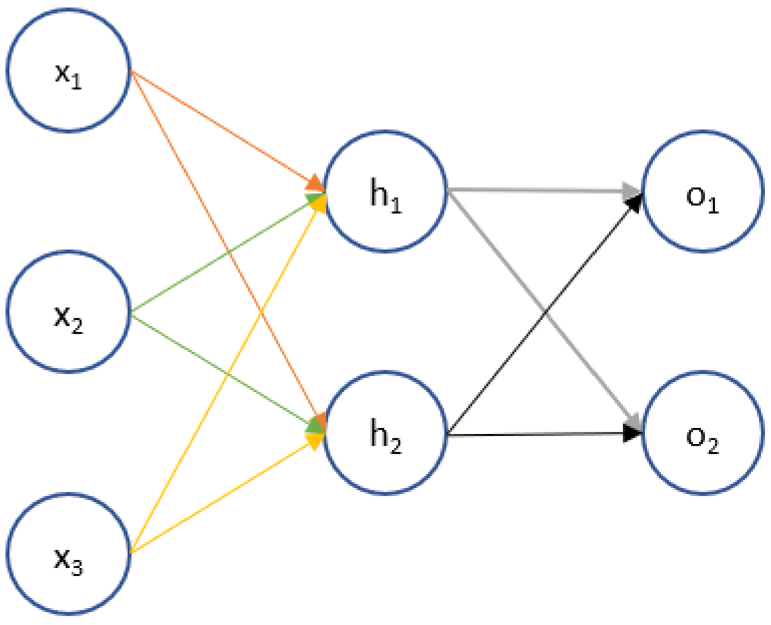
BACKPROPAGATION EXAMPLE WITH NUMBERS STEP BY STEP

POSTED ON FEBRUARY 28, 2019 BY ADMIN

When I come across a new mathematical concept or before I use a canned software package, I like to replicate the calculations in order to get a deeper understanding of what is going on. This type of computation based approach from first principles helped me greatly when I first came across material on artificial neural networks.

In this post, I go through a detailed example of one iteration of the backpropagation algorithm using full formulas from basic principles and actual values. The neural network I use has three input neurons, one hidden layer with two neurons, and an output layer with two neurons.



The following are the (very) high level steps that I will take in this post. Details on each step will follow after.

(1) Initialize weights for the parameters we want to train

(2) Forward propagate through the network to get the output values

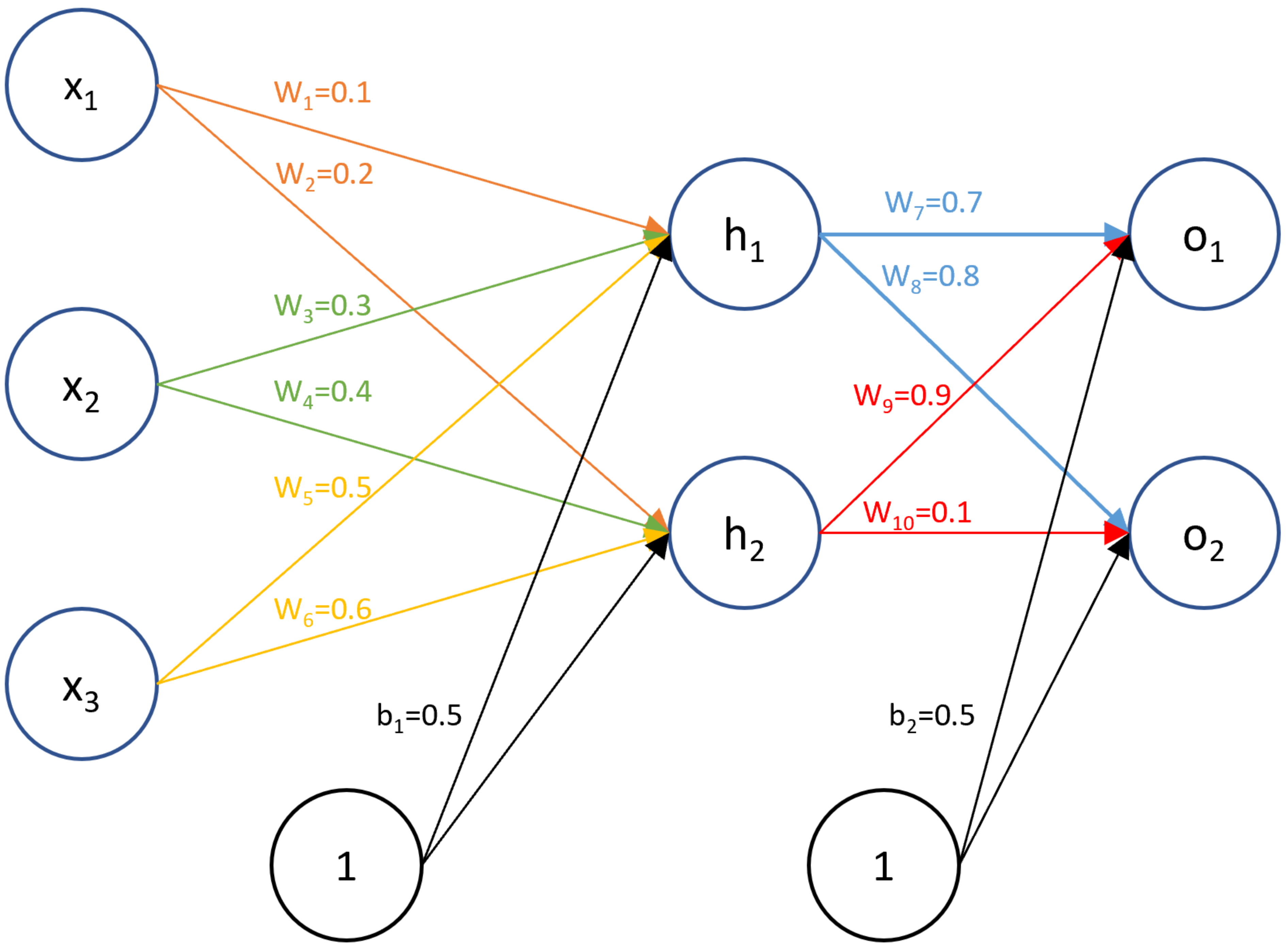
(3) Define the error or cost function and its first derivatives

(4) Backpropagate through the network to determine the error derivatives

(5) Update the parameter estimates using the error derivative and the current value

Step 1

The input and target values for this problem are $x_1 = 1, x_2 = 4, x_3 = 5$ and $t_1 = 0.1, t_2 = 0.05$. I will initialize weights as shown in the diagram below. Generally, you will assign them randomly but for illustration purposes, I've chosen these numbers.



Step 2

Mathematically, we have the following relationships between nodes in the networks. For the input and output layer, I will use the somewhat strange convention of denoting z_{h_1}, z_{h_2}, z_{o_1}, and z_{o_2} to denote the value before the activation function is applied and the notation of h_1, h_2, o_1, and o_2 to denote the values after application of the activation function.

Input to hidden layer

$$w_1x_1 + w_3x_2 + w_5x_3 + b_1 = z_{h_1}$$

$$w_2x_1 + w_4x_2 + w_6x_3 + b_1 = z_{h_2}$$

$$h_1 = \sigma(z_{h_1})$$

$$h_2 = \sigma(z_{h_2})$$

Hidden layer to output layer

$$w_7h_1 + w_9h_2 + b_2 = z_{o_1}$$

$$w_8h_1 + w_{10}h_2 + b_2 = z_{o_2}$$

$$o_1 = \sigma(z_{o_1})$$

$$o_2 = \sigma(z_{o_2})$$

We can use the formulas above to forward propagate through the network. I've shown up to four decimal places below but maintained all decimals in actual calculations.

$$w_1x_1 + w_3x_2 + w_5x_3 + b_1 = z_{h_1} = 0.1(1) + 0.3(4) + 0.5(5) + 0.5 = 4.3$$
$$h_1 = \sigma(z_{h_1}) = \sigma(4.3) = 0.9866$$

$$w_2x_1 + w_4x_2 + w_6x_3 + b_1 = z_{h_2} = 0.2(1) + 0.4(4) + 0.6(5) + 0.5 = 5.3$$
$$h_2 = \sigma(z_{h_2}) = \sigma(5.3) = 0.9950$$

$w_7 h_1 + w_9 h_2 + b_2 = z_{o_1} = 0.7(0.9866) + 0.9(0.9950) + 0.5 = 2.0862$
$o_1 = \sigma(z_{o_1}) = \sigma(2.0862) = 0.8896$

$w_8 h_1 + w_{10} h_2 + b_2 = z_{o_2} = 0.8(0.9866) + 0.1(0.9950) + 0.5 = 1.3888$
$o_2 = \sigma(z_{o_2}) = \sigma(1.3888) = 0.8004$

Step 3

We now define the sum of squares error using the target values and the results from the last layer from forward propagation.

$E = \frac{1}{2}[(o_1 - t_1)^2 + (o_2 - t_2)^2$

$\frac{dE}{d_{o_1}} = o_1 - t_1$

$\frac{dE}{d_{o_2}} = o_2 - t_2$

Step 4

We are now ready to backpropagate through the network to compute all the error derivatives with respect to the parameters. Note that although there will be many long formulas, we are not doing anything fancy here. We are just using the basic principles of calculus such as the chain rule.

First we go over some derivatives we will need in this step. The derivative of the sigmoid function is given here. Also, given that $w_7 h_1 + w_9 h_2 + b_2 = z_{o_1}$ and $w_8 h_1 + w_{10} h_2 + b_2 = z_{o_2}$, we have $\frac{dz_{o_1}}{dw_7} = h_1$, $\frac{dz_{o_2}}{dw_8} = h_1$, $\frac{dz_{o_1}}{dw_9} = h_2$, $\frac{dz_{o_2}}{dw_{10}} = h_2$, $\frac{dz_{o_1}}{db_2} = 1$, and $\frac{dz_{o_2}}{db_2} = 1$.

We are now ready to calculate $\frac{dE}{dw_7}$, $\frac{dE}{dw_8}$, $\frac{dE}{dw_9}$, and $\frac{dE}{dw_{10}}$ using the derivatives we have already discussed.

$\frac{dE}{dw_7} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{dw_7}$

$\frac{dE}{dw_7} = (o_1 - t_1)(o_1(1 - o_1))h_1$

$\frac{dE}{dw_7} = (0.8896 - 0.1)(0.8896(1 - 0.8896))(0.9866)$

$\frac{dE}{dw_7} = 0.0765$

I will omit the details on the next three computations since they are very similar to the one above. Feel free to leave a comment if you are unable to replicate the numbers below.

$\frac{dE}{dw_8} = \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{dw_8}$

$\frac{dE}{dw_8} = (0.7504)(0.1598)(0.9866)$

$\frac{dE}{dw_8} = 0.1183$

$\frac{dE}{dw_9} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{dw_9}$

$\frac{dE}{dw_9} = (0.7896)(0.0983)(0.9950)$

$\frac{dE}{dw_9} = 0.0772$

$\frac{dE}{dw_{10}} = \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{dw_{10}}$

$\frac{dE}{dw_{10}} = (0.7504)(0.1598)(0.9950)$

$\frac{dE}{dw_{10}} = 0.1193$

The error derivative of b_2 is a little bit more involved since changes to b_2 affect the error through both o_1 and o_2.

$\frac{dE}{db_2} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{db_2} + \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{db_2}$

$\frac{dE}{db_2} = (0.7896)(0.0983)(1) + (0.7504)(0.1598)(1)$

$\frac{dE}{db_2} = 0.1975$

To summarize, we have computed numerical values for the error derivatives with respect to w_7, w_8, w_9, w_{10}, and b_2. We will now backpropagate one layer to compute the error derivatives of the parameters connecting the input layer to the hidden layer. These error derivatives are $\frac{dE}{dw_1}$, $\frac{dE}{dw_2}$, $\frac{dE}{dw_3}$, $\frac{dE}{dw_4}$, $\frac{dE}{dw_5}$, $\frac{dE}{dw_6}$, and $\frac{dE}{db_1}$.

I will calculate $\frac{dE}{dw_1}$, $\frac{dE}{dw_3}$, and $\frac{dE}{dw_5}$ first since they all flow through the h_1 node.

$\frac{dE}{dw_1} = \frac{dE}{dh_1}\frac{dh_1}{dz_{h_1}}\frac{dz_{h_1}}{dw_1}$

The calculation of the first term on the right hand side of the equation above is a bit more involved than previous calculations since h_1 affects the error through both o_1 and o_2.

$\frac{dE}{dh_1} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{dh_1} + \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{dh_1}$

Now I will proceed with the numerical values for the error derivatives above. These derivatives have already been calculated above or are similar in style to those calculated above. If anything is unclear, please leave a comment.

$\frac{dE}{dh_1} = (0.7896)(0.0983)(0.7) + (0.7504)(0.1598)(0.8) = 0.1502$

Plugging the above into the formula for $\frac{dE}{dw_1}$, we get

$\frac{dE}{dw_1} = (0.1502)(0.0132)(1) = 0.0020$

The calculations for $\frac{dE}{dw_3}$ and $\frac{dE}{dw_5}$ are below

$\frac{dE}{dw_3} = \frac{dE}{dh_1}\frac{dh_1}{dz_{h_1}}\frac{dz_{h_1}}{dw_3}$

$\frac{dE}{dw_3} = (0.1502)(0.0132)(4) = 0.0079$

$\frac{dE}{dw_5} = \frac{dE}{dh_1}\frac{dh_1}{dz_{h_1}}\frac{dz_{h_1}}{dw_5}$

$\frac{dE}{dw_5} = (0.1502)(0.0132)(5) = 0.0099$

I will now calculate $\frac{dE}{dw_2}$, $\frac{dE}{dw_4}$, and $\frac{dE}{dw_6}$ since they all flow through the h_2 node.

$\frac{dE}{dw_2} = \frac{dE}{dh_2}\frac{dh_2}{dz_{h_2}}\frac{dz_{h_2}}{dw_2}$

The calculation of the first term on the right hand side of the equation above is a bit more involved since h_2 affects the error through both o_1 and o_2.

$\frac{dE}{dh_2} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{dh_2} + \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{dh_2}$

$\frac{dE}{dh_2} = (0.7896)(0.0983)(0.9) + (0.7504)(0.1598)(0.1) = 0.0818$

Plugging the above into the formula for $\frac{dE}{dw_2}$, we get

$\frac{dE}{dw_2} = (0.0818)(0.0049)(1) = 0.0004$

The calculations for $\frac{dE}{dw_4}$ and $\frac{dE}{dw_6}$ are below

$\frac{dE}{dw_4} = \frac{dE}{dh_2}\frac{dh_2}{dz_{h_2}}\frac{dz_{h_2}}{dw_4}$

$\frac{dE}{dw_4} = (0.0818)(0.0049)(4) = 0.0016$

$\frac{dE}{dw_6} = \frac{dE}{dh_2}\frac{dh_2}{dz_{h_2}}\frac{dz_{h_2}}{dw_6}$

$\frac{dE}{dw_6} = (0.0818)(0.0049)(5) = 0.0020$

The final error derivative we have to calculate is $\frac{dE}{db_1}$, which is done next

$\frac{dE}{db_1} = \frac{dE}{do_1}\frac{do_1}{dz_{o_1}}\frac{dz_{o_1}}{dh_1}\frac{dh_1}{dz_{h_1}}\frac{dz_{h_1}}{db_1} + \frac{dE}{do_2}\frac{do_2}{dz_{o_2}}\frac{dz_{o_2}}{dh_2}\frac{dh_2}{dz_{h_2}}\frac{dz_{h_2}}{db_1}$

$\frac{dE}{db_1} = (0.7896)(0.0983)(0.7)(0.0132)(1) + (0.7504)(0.1598)(0.1)(0.0049)(1) = 0.0008$

We now have all the error derivatives and we're ready to make the parameter updates after the first iteration of backpropagation. We will use the learning rate of $\alpha = 0.01$

$$w_1 := w_1 - \alpha\frac{dE}{dw_1} = 0.1 - (0.01)(0.0020) = 0.1000$$

$$w_2 := w_2 - \alpha\frac{dE}{dw_2} = 0.2 - (0.01)(0.0004) = 0.2000$$

$$w_3 := w_3 - \alpha\frac{dE}{dw_3} = 0.3 - (0.01)(0.0079) = 0.2999$$

$$w_4 := w_4 - \alpha\frac{dE}{dw_4} = 0.4 - (0.01)(0.0016) = 0.4000$$

$$w_5 := w_5 - \alpha\frac{dE}{dw_5} = 0.5 - (0.01)(0.0099) = 0.4999$$

$$w_6 := w_6 - \alpha \frac{dE}{dw_6} = 0.6 - (0.01)(0.0020) = 0.6000$$

$$w_7 := w_7 - \alpha \frac{dE}{dw_7} = 0.7 - (0.01)(0.0765) = 0.6992$$

$$w_8 := w_8 - \alpha \frac{dE}{dw_8} = 0.8 - (0.01)(0.1183) = 0.7988$$

$$w_9 := w_9 - \alpha \frac{dE}{dw_9} = 0.9 - (0.01)(0.0772) = 0.8992$$

$$w_{10} := w_{10} - \alpha \frac{dE}{dw_{10}} = 0.1 - (0.01)(0.1193) = 0.0988$$

$$b_1 := b_1 - \alpha \frac{dE}{db_1} = 0.5 - (0.01)(0.0008) = 0.5000$$

$$b_2 := b_2 - \alpha \frac{dE}{db_2} = 0.5 - (0.01)(0.1975) = 0.4980$$

So what do we do now? We repeat that over and over many times until the error goes down and the parameter estimates stabilize or converge to some values. We obviously won't be going through all these calculations manually. I've provided Python code below that codifies the calculations above. Nowadays, we wouldn't do any of these manually but rather use a machine learning package that is already readily available.

In [1]:
```
import numpy as np
import pandas as pd
%matplotlib inline
```

In [2]:
```
numIter = 10000

# Initialize Variables
w1 = 0.1
w2 = 0.2
w3 = 0.3
w4 = 0.4
w5 = 0.5
w6 = 0.6
w7 = 0.7
w8 = 0.8
w9 = 0.9
w10 = 0.1
wList = [w1, w2, w3, w4, w5, w6, w7, w8, w9, w10]

b1 = 0.5
b2 = 0.5
bList = [b1, b2]

# Input and Target values
x1 = 1
x2 = 4
x3 = 5
xList = [x1, x2, x3]

t1 = 0.1
t2 = 0.05
tList = [t1, t2]

# set learning rate
alpha = 0.01
```

In [3]:
```
def sigmoid(x):
    return np.divide(1, (1 + np.exp(-x)))
```

In [4]:
```
def forwardProp(xList, wList, bList):
    zh1 = wList[0] * xList[0] + wList[2] * xList[1] + wList[4] * xList[2] + bList[0]
    zh2 = wList[1] * xList[0] + wList[3] * xList[1] + wList[5] * xList[2] + bList[0]
    h1 = sigmoid(zh1)
    h2 = sigmoid(zh2)
    zo1 = wList[6] * h1 + wList[8] * h2 + bList[1]
    zo2 = wList[7] * h1 + wList[9] * h2 + bList[1]
    o1 = sigmoid(zo1)
    o2 = sigmoid(zo2)
    return h1, h2, o1, o2
```

In [5]:
```
def error(oList, tList):
    return 0.5 * (np.power(oList[0] - tList[0], 2) + np.power(oList[1] - tList[1], 2))
```

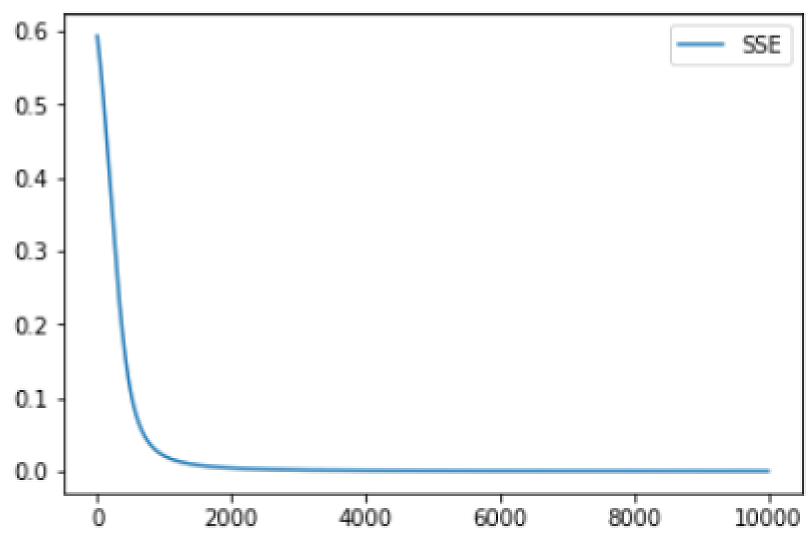
In [6]:

```
errList = []
for i in range(numIter):
    # Forward propagation
    h1, h2, o1, o2 = forwardProp(xList, wList, bList)

    # Compute Error
    sse = error([o1, o2], tList)
    errList.append(sse)

    print('Running ' + str(i + 1) + ' of ' + str(numIter))
    print('o1: ' + str(o1))
    print('t1: ' + str(t1))
    print('o2: ' + str(o2))
    print('t2: ' + str(t2))
    print('error: ' + str(sse))
    print('')

    # Error derivative calculations
    # Compute dE_dw7
    dE_do1 = o1 - t1
    do1_dzo1 = o1 * (1-o1)
    dzo1_dw7 = h1
    dE_dw7 = dE_do1 * do1_dzo1 * dzo1_dw7
    # Compute dE_dw8
    dE_do2 = o2 - t2
    do2_dzo2 = o2 * (1 - o2)
    dzo2_dw8 = h1
    dE_dw8 = dE_do2 * do2_dzo2 * dzo2_dw8
    # Compute dE_dw9
    dzo1_dw9 = h2
    dE_dw9 = dE_do1 * do1_dzo1 * dzo1_dw9
    # Compute dE_dw10
    dzo2_dw10 = h2
    dE_dw10 = dE_do2 * do2_dzo2 * dzo2_dw10
```

```
    # Compute dE_db2
    dzo1_db2 = 1
    dzo2_db2 = 1
    dE_db2 = dE_do1 * do1_dzo1 * dzo1_db2 + dE_do2 * do2_dzo2 * dzo2_db2
    # Compute dE_dh1 first
    dzo1_dh1 = w7
    dzo2_dh1 = w8
    dE_dh1 = dE_do1 * do1_dzo1 * dzo1_dh1 + dE_do2 * do2_dzo2 * dzo2_dh1
    # Compute dE_dw1
    dh1_dzh1 = h1 * (1 - h1)
    dzh1_dw1 = x1
    dE_dw1 = dE_dh1 * dh1_dzh1 * dzh1_dw1
    # Compute dE_dw3
    dzh1_dw3 = x2
    dE_dw3 = dE_dh1 * dh1_dzh1 * dzh1_dw3
    # Compute dE_dw5
    dzh1_dw5 = x3
    dE_dw5 = dE_dh1 * dh1_dzh1 * dzh1_dw5
    # Compute dE_dh2 first
    dzo1_dh2 = w9
    dzo2_dh2 = w10
    dE_dh2 = dE_do1 * do1_dzo1 * dzo1_dh2 + dE_do2 * do2_dzo2 * dzo2_dh2
    # Compute dE_dw2
    dh2_dzh2 = h2 * (1 - h2)
    dzh2_dw2 = x1
    dE_dw2 = dE_dh2 * dh2_dzh2 * dzh2_dw2
    # Compute dE_dw4
    dzh2_dw4 = x2
    dE_dw4 = dE_dh2 * dh2_dzh2 * dzh2_dw4
    # Compute dE_dw6
    dzh2_dw6 = x3
    dE_dw6 = dE_dh2 * dh2_dzh2 * dzh2_dw6
    # Compute dE_db1
    dzh1_db1 = 1
    dzh2_db1 = 1
    term1 = dE_do1 * do1_dzo1 * dzo1_dh1 * dh1_dzh1 * dzh1_db1
    term2 = dE_do2 * do2_dzo2 * dzo2_dh2 * dh2_dzh2 * dzh2_db1
    dE_db1 = term1 + term2
    # Update all parameters
    w1 = w1 - alpha * dE_dw1
    w2 = w2 - alpha * dE_dw2
    w3 = w3 - alpha * dE_dw3
    w4 = w4 - alpha * dE_dw4
    w5 = w5 - alpha * dE_dw5
    w6 = w6 - alpha * dE_dw6
    w7 = w7 - alpha * dE_dw7
    w8 = w8 - alpha * dE_dw8
    w9 = w9 - alpha * dE_dw9
    w10 = w10 - alpha * dE_dw10
    b1 = b1 - alpha * dE_db1
    b2 = b2 - alpha * dE_db2
    wList = [w1, w2, w3, w4, w5, w6, w7, w8, w9, w10]
    bList = [b1, b2]
```

I ran 10,000 iterations and we see below that sum of squares error has dropped significantly after the first thousand or so iterations.

```
In [7]:  pd.DataFrame(errList, columns=['SSE']).plot()

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1b6cad97668>
```



POSTED IN MACHINE LEARNING TAGGED AI, ANN, ARTIFICIAL INTELLIGENCE, BACKPROPAGATION, FORWARD PROPAGATION, MACHINE LEARNING, NEURAL NETWORK

← Website will be Restored Soon

Algorithmic Trading Day 1 – Introduction →

9 THOUGHTS ON "BACKPROPAGATION EXAMPLE WITH NUMBERS STEP BY STEP"

jpowersbaseball says:

DECEMBER 30, 2019 AT 5:28 PM

When I use gradient checking to evaluate this algorithm, I get some odd results. For instance, w5's gradient calculated above is 0.0099. But when I calculate the costs of the network when I adjust w5 by 0.0001 and -0.0001, I get 3.5365879 and 3.5365727 whose difference divided by 0.0002 is 0.07614, 7 times greater than the calculated gradient. I think I'm doing my checking correctly?

REPLY

Sebastian says:

DECEMBER 26, 2019 AT 2:12 AM

Wow this was incredibly clear thanks.

How would other observations be incorporated into the back-propagation though? From this process it seems like all you need is one vector of input values.

REPLY

mICON says:

DECEMBER 4, 2019 AT 10:19 AM

And also:

(0.7896 * 0.0983 * 0.7 * 0.0132 * 1) + (0.7504 * 1598 * 0.1 * 0.0049 * 1);
-> 0.5882953953632 not 0.0008

REPLY

mICON says:

DECEMBER 4, 2019 AT 10:20 AM

My bad, forget a 0. there

REPLY

mICON says:

DECEMBER 3, 2019 AT 3:04 PM

In your final calculation of db1, you chain derivates from w7 and w10, not w8 and w9, why?

REPLY

kyunghoon says:

MARCH 20, 2019 AT 5:22 AM

nevermind, figured it out, you meant for t2 to equal .05 not .5.

thanks for the write-up.

REPLY

kyunghoon says:

MARCH 20, 2019 AT 5:16 AM

how are you computing dE/do2 = .7504?

you state:
dE/do2 = o2 – t2
o2 = .8004
t2 = .5

therefore:
dE/do2 = (.8004) – (.5) = .3004 (not .7504)

am i missing something?

REPLY

Anonymous says:

JANUARY 21, 2020 AT 9:41 PM

t2 = 0.05 not 0.6

REPLY

Ganesh Chandra Satish says:

MARCH 18, 2019 AT 12:06 AM

Thanks for the post. It explained backprop perfectly.

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Message...

Name * | Email * | Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment

SPONSORED

RECENT POSTS

- A Comparison of COVID-19 Vaccine Stocks
- A Comparison of Gold ETFs
- Algorithmic Trading Day 8 – Performance on Different Currency Pairs
- Understanding and Replicating LSTM Networks
- Reading List

SPONSORED

CATEGORIES

- Algorithmic Trading
- Comparisons within Sectors
- Decision Trees
- Deep Neural Networks
- Forex
- LSTM
- Machine Learning
- OANDA
- Random Thoughts
- Toronto Real Estate
- Uncategorized

SPONSORED

ARCHIVES

- August 2020
- July 2020
- May 2020
- April 2020
- March 2020
- December 2019
- November 2019
- May 2019

- April 2019
- March 2019
- February 2019

CUSTOM