Matt Mazur

Menu

Skip to content

- Home
- About
- Archives
- Contact
- Projects

A Step by Step Backpropagation Example

Background

Backpropagation is a common method for training a neural network. There is no shortage of papers online that attempt to explain how backpropagation works, but few that include an example with actual numbers. This post is my attempt to explain how it works with a concrete example that folks can compare their own calculations to in order to ensure they understand backpropagation correctly.

Backpropagation in Python

You can play around with a Python script that I wrote that implements the backpropagation algorithm in this Github repo.

Backpropagation Visualization

For an interactive visualization showing a neural network as it learns, check out my Neural Network visualization.

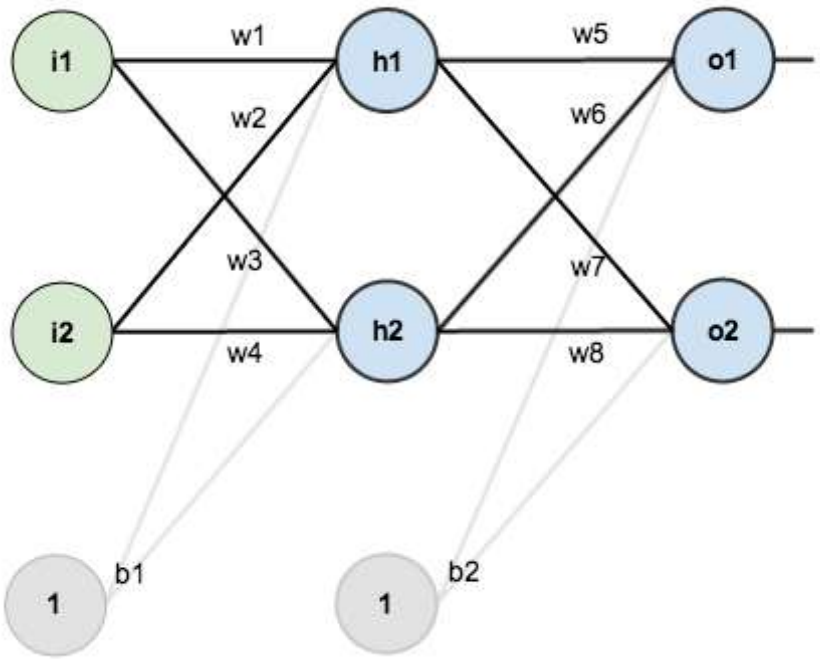
Additional Resources

If you find this tutorial useful and want to continue learning about neural networks, machine learning, and deep learning, I highly recommend checking out Adrian Rosebrock's new book, Deep Learning for Computer Vision with Python. I really enjoyed the book and will have a full review up soon.

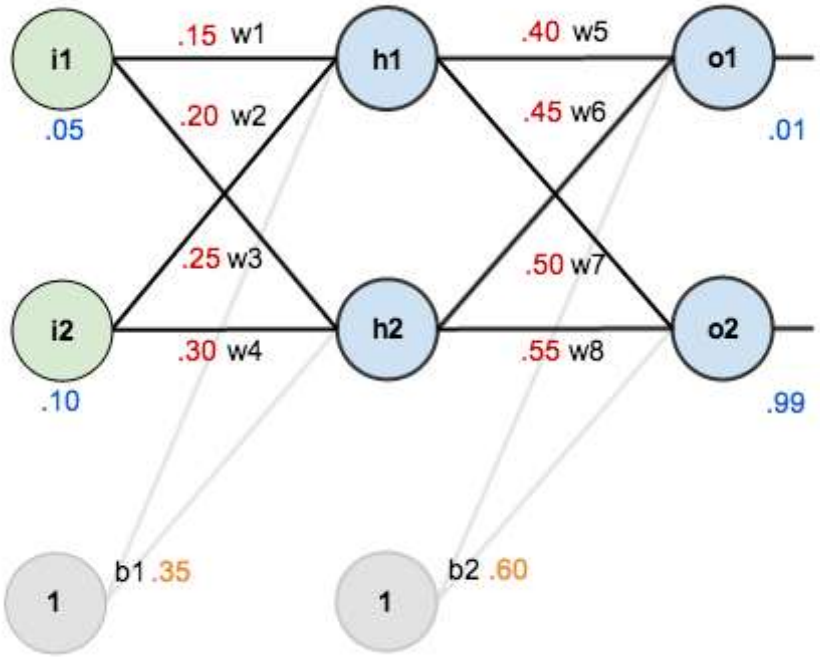
Overview

For this tutorial, we're going to use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.

Here's the basic structure:



In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

The Forward Pass

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network.

We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.

Total net input is also referred to as just *net input* by some sources.

Here's how we calculate the total net input for h_1:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We then squash it using the logistic function to get the output of h_1:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here's the output for o_1:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for o_2 we get:

$$out_{o2} = 0.772928465$$

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Some sources refer to the target as the *ideal* and the output as the *actual*.

The $\frac{1}{2}$ is included so that exponent is cancelled when we differentiate later on. The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here [1].

For example, the target output for o_1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

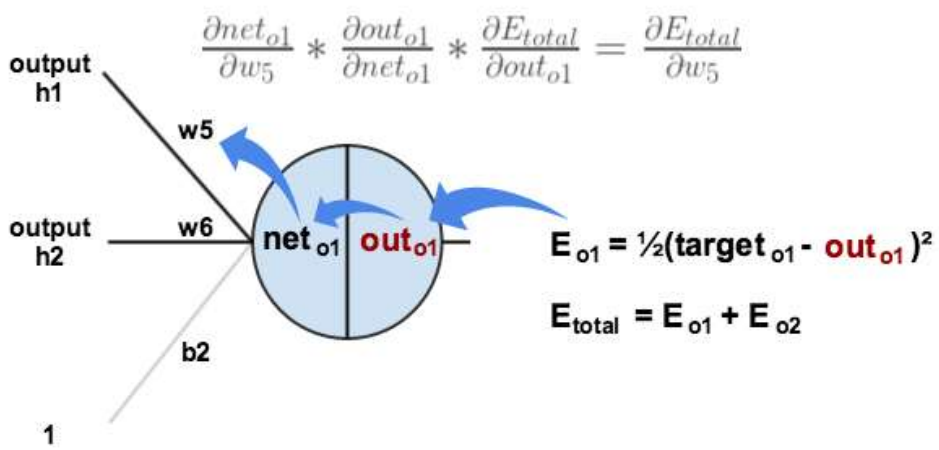
Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Visually, here's what we're doing:



We need to figure out each piece in this equation.

First, how much does the total error change with respect to the output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$-(target - out)$ is sometimes expressed as $out - target$
When we take the partial derivative of the total error with respect to out_{o1}, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative of a constant which is zero.

Next, how much does the output of o_1 change with respect to its total net input?

The partial derivative of the logistic function is the output multiplied by 1 minus the output:

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Finally, how much does the total net input of $o1$ change with respect to w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Some sources use α (alpha) to represent the learning rate, others use η (eta), and others even use ϵ (epsilon).

We can repeat this process to get the new weights w_6, w_7, and w_8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, and w_4.

Big picture, here's what we need to figure out:

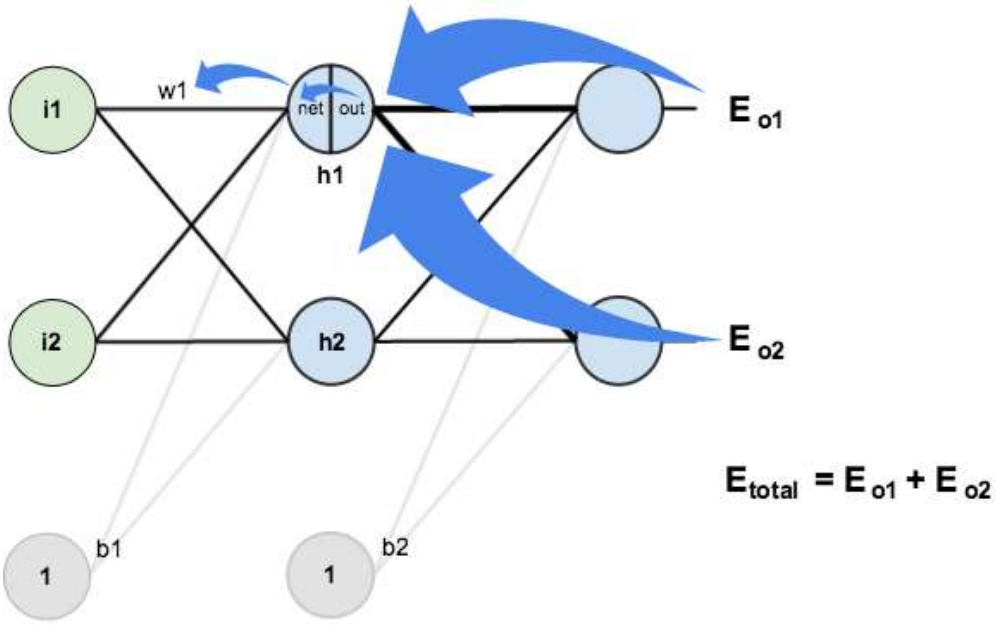
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Visually:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$

Therefore:

$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$

$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$

$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

Putting it all together:

$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$

$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$

You might also see this written as:

$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$

$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \delta_o * w_{ho}) * out_{h1}(1 - out_{h1}) * i_1$

$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$

We can now update w_1:

$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$

Repeating this for w_2, w_3, and w_4

$w_2^+ = 0.19956143$

$w_3^+ = 0.24975114$

$w_4^+ = 0.29950229$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feed

forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

If you've made it this far and found any errors in any of the above or can think of any ways to make it clearer for future readers, don't hesitate to drop me a note. Thanks!

Share this:

- Twitter
- Facebook
-

Like this:

152 bloggers like this.

Related

Experimenting with a Neural Network-based Poker Bot
October 13, 2009
In "Poker Bot"

The State of Emergent Mind
December 11, 2015
In "Emergent Mind"

I'm going to write more often. For real this time.
November 25, 2015
In "Writing"

Posted on March 17, 2015 by Mazur. This entry was posted in Machine Learning and tagged ai, backpropagation, machine learning, neural networks. Bookmark the permalink.

Post navigation

← Introducing ABTestCalculator.com, an Open Source A/B Test Significance Calculator
TetriNET Bot Source Code Published on Github →

993 thoughts on "A Step by Step Backpropagation Example"

Comment navigation

← Older Comments

1. Как устроена нейросеть / Блог компании BCS FinTech / Хабр

2. *Matt* says:
   July 8, 2020 at 10:28 am

   Thanks for this nice illustration of backpropagation!
   I am wondering how the calculations must be modified if we have more than 1 training sample data (e.g. 2 samples).

   Reply

   - *Matt* says:
     October 26, 2020 at 7:45 pm

     I reply to myself… I forgot to apply the chainrule

     Reply

3. *Shahrum* says:
   July 9, 2020 at 10:50 am

   It seems that you have totally forgotten to update b1 and b2! They are part of the weights (parameters) of the network. Or am I missing something here?

   Reply

   - *Abhimalya Chowdhury* says:
     September 7, 2020 at 1:44 am

     We never update bias. Refer Andrew Ng's Machine Learning course on coursera

     Reply

     - *L.M.Dao* says:
       November 5, 2020 at 10:06 am

       I think this is not the case
       https://stackoverflow.com/questions/3775032/how-to-update-the-bias-in-neural-network-backpropagation

       Reply

4. *Seong* says:
   July 20, 2020 at 2:33 am

   Thanks to your nice illustration, now I've understood backpropagation.

   Reply

5. *deva* says:
   July 23, 2020 at 9:14 pm

This is exactly what i was needed , great job sir, super easy explanation.

Reply

6.  *pawandtu@gmail.com* says:
August 3, 2020 at 1:59 pm

Just wondering about the range of the learning rate. Why can't it be greater than 1?

Reply

7.  *ANIL BABU B* says:
August 8, 2020 at 11:19 am

Hi Matt
Thanks for giving the link, but i have following queries, can you please clarify
1. Why bias weights are not updated anywhere
2.Outputs at hidden and Output layers are not independent of the initial weights chosen at the input layer. So for calculated optimal weights at input layer (w1 to w4) why final Etot is again differentiated w.r.t w1, instead should we not calculate the errors at the hidden layer using the revised weights of w5 to w8 and then use the same method for calculating revised weights w1 to w4 by differentiating this error at hidden layer w.r.t w1.
3.Error at hidden layer can be calculated as follows: We already know the out puts at the hidden layer in forward propagation , these we will take as initial values, then using the revised weights of w5 to w8 we will back calculate the revised outputs at hidden layer, the difference we can take as errors
4. i calculated the errors as mentioned in step 3, i got the outputs at h1 and h2 are -3.8326165 and 4.6039905. Since these are outputs at hidden layer , these are outputs of sigmoid function so values should always be between 0 and 1, but the values here are outside the outputs of sigmoid function range

Please clarify why and where the flaw is

Reply

-  *Harrison* says:
February 1, 2021 at 5:54 am

Question number two is a good one. DId you get an answer yet?

Reply

-  *Neel Joshi* says:
April 6, 2021 at 6:46 am

Answer for Q.2, 3 and 4: We do not have ground truth (correct answers) available for hidden layer neurons. So, we can not calculate errors as you have mentioned in point no.3. Error can be calculated only if we have correct answers.

There is not any meaning in doing the same. The values we are getting at hidden layer are due to initial weights (which were randomly guessed). So, these are value are also random guesses.

That's why we consider Etot only to update all the weights.

Reply

-  *Divyanshu Agarwal* says:
  April 21, 2021 at 10:07 am

  Part 2.
  Suppose you have a function
  y = exp(z)
  z = x^2
  dy/dx = (dy/dz) * (dz/dx) = (exp(x^2))*2x
  think of y is is the function you are optimizing and x is input.
  now to calculate gradient with respect to input what we do in backpropagation is we first calculate dz/dx which is 2x (after which we update weights for next forward pass).Then we calculate dy/dz and then we use chain rule to get dy/dx.
  If you calculate dy/dz by updated weights you will not get correct gradient .

  Reply

8.  *Clara Liu* says:
   August 14, 2020 at 6:30 am

   Awesome tutorial!

   But are there possibly calculation errors for the undemonstrated weights? I kept getting slightly different updated weight values for the hidden layer…

   But let's take a simpler one for example:
   For dEtotal/dw7, the calculation should be very similar to dEtotal/dw5, by just changing the last partial derivative to dnet o1/dw7, which is essentially out h2.So dEtotal/dw7 = 0.74136507*0.186815602*0.596884378 = 0.08266763

   new w7 = 0.5-(0.5*0.08266763)= 0.458666185.

   But your answer is 0.511301270…

   Perhaps I made a mistake in my calculation? Some clarification would be great!

   Reply

   -  *Albrecht Ehlert* says:
     October 7, 2020 at 1:02 am

     0.5113012702387375 is right …

Reply

-  *AJ* says:
  November 26, 2020 at 1:26 am

  The number you have there, 0.08266763, is actually dEtotal/dw6. You will see that applying it to the original w6 yields the value he gave: 0.45 – (0.5*0.08266763) = 0.40866186.

  Maybe you confused w7 and w6? The diagram makes it easy to confuse them. W7 is the weight between h1 and o2.

  To find dEtotal/dw7 you would have to find:
  dEtotal/dout_{o_2} * dout_{o_2}/dnet_{o_2} * dnet_{o_2}/dw7.

  so dEtotal/dw7 = -0.21707153 * 0.17551005 * 0.59326999 = -0.02260254

  new w7 = 0.5 – (0.5 * -0.02260254) = 0.511301270
  hope this helped

  Reply

9. Vectorization of Neural Nets | My Universal NK

10.  *Luke* says:
    September 13, 2020 at 8:48 am

    After many hours of looking for a resource that can efficiently and clearly explain math behind backprop, I finally found it! Fantastic work!

    Reply

11.  *Joshua Laferriere* says:
    September 18, 2020 at 7:36 pm

    Heaton in his book on neural networks math say
    node deltas are based on [sum] “sum is for derivatives, output is for gradient, else your applying the activation function twice?”

    but I’m starting to question his book because he also applies derivatives to the sum

    “Ii is important to note that in the above equation, we are multiplying by the output of hidden I. not the sum. When dealing directly with a derivative you should supply the sum Otherwise, you would be indirectly applying the activation function twice.”

    but I see your example and one more where that’s not the case
    https://github.com/thistleknot/Ann-v2/blob/master/myNueralNet.cpp

    I see two examples where the derivative is applied to the output

Reply

12. *Anam Anwar* says:
September 21, 2020 at 10:47 am

Very well explained…… Really helped alot in my final exams….. Thanks

Reply

13. Neural Network – Bagus's digital notes

14. *Albrecht Ehlert* says:
October 8, 2020 at 1:55 am

Dear Matt,
thank you for the nice illustration!
I built the network and get exactly your outputs:

Weights and Bias of Hidden Layer:
Neuron 1: 0.1497807161327628 0.19956143226552567 0.35
Neuron 2: 0.24975114363236958 0.29950228726473915 0.35

Weights and Bias of Output Layer:
Neuron 1: 0.35891647971788465 0.4086661860762334 0.6
Neuron 2: 0.5113012702387375 0.5613701211079891 0.6

output:
0.7513650695523157 0.7729284653214625

Than I made a experiment with the bias. Without changing the bias I got after 1000 epoches the following outputs:

Weights and Bias of Hidden Layer:
Neuron 1: 0.2820419392605305 0.4640838785210599 0.35
Neuron 2: 0.3805890849512254 0.5611781699024483 0.35

Weights and Bias of Output Layer:
Neuron 1: -3.0640975297007556 -3.034730378052809 0.6
Neuron 2: 2.0517051904569836 2.110885730396752 0.6

output:
0.044075530730776365 0.9572825838174545

With backpropagation of the bias the outputs getting better:

Weights and Bias of Hidden Layer:
Neuron 1: 0.20668916041682514 0.3133783208336505 1.4753841161727905
Neuron 2: 0.3058492464890622 0.4116984929781265 1.4753841161727905

Weights and Bias of Output Layer:
Neuron 1: -2.0761119815104956 -2.038231681376019 -0.08713942766189575
Neuron 2: 2.137631425033325 2.194909264537856 -0.08713942766189575

output:
0.03031757858059988 0.9698293077608338

Sincerly
Albrecht Ehlert from Germany

Reply

- *Hello$* says:
  January 19, 2021 at 9:14 am

  Could you please share the calculation I mean, how do we update the bias b1 and b2 when we do backpropagation.

  Reply

  - *Hello$* says:
    January 19, 2021 at 9:15 am

    @Albrecht Ehlert Could you please share the calculation I mean, how do we update the bias b1 and b2 when we do backpropagation.

    Reply

  - *Albrecht Ehlert* says:
    January 29, 2021 at 11:53 am

    Hi Hello§,

    here is the output of my program:

    ————————————————————
    weights and bias of hidden layer:
    neuron 1: 0.15 0.2 0.35
    neuron 2: 0.25 0.3 0.35

    weights and bias of output layer:
    neuron 1: 0.4 0.45 0.6
    neuron 2: 0.5 0.55 0.6

    output 1: 0.7513650695523157 0.7729284653214625

    average of total errors: 0.2983711087600027

    learned in percent: 0

    epoche: 1 number of data: 1

output:
0.7513650695523157 0.7729284653214625

total error before: 0.2983711087600027

backpropagation output layer

output: 0.7513650695523157 target: 0.01

new weight: 0.35891647971788465
new weight: 0.4086661860762334
bias calculation 1: 0.5307507191857215

output: 0.7729284653214625 target: 0.99

new weight: 0.5113012702387375
new weight: 0.5613701211079891
bias calculation 2: 0.6190491182582781

backpropagation hidden layer

weight adjustment: -1.8236143879704674E-4
new weight: 0.14981763856120295
weight adjustment: -3.647228775940935E-4
new weight: 0.19963527712240592
bias calculation 1: 0.34635277122405905
weight adjustment: -2.1181480223381405E-4
new weight: 0.2497881851977662
weight adjustment: -4.236296044676281E-4
new weight: 0.29957637039553237
bias calculation 2: 0.3457637039553237

————————————————————————

You calculate the bias like a regular weight.
But you get for b2 two results:
bias calculation 1: 0,5307507191857215
bias calculation 2: 0.6190491182582781
The program takes the average of both as the new b2: 0,5748999187219998

The same with b1:
bias calculation 1: 0.34635277122405905
bias calculation 2: 0.3457637039553237
The program takes the average of both as the new b1: 0,3460582375896914

Reply

15. 한성주 says:
October 17, 2020 at 10:27 pm

I finally understood BP thanks to you. You are my hero.

Reply

16.  *Matt* says:
October 26, 2020 at 7:36 am

When you derive E_total for out_o1 could you please explain where the -1 comes from? I get the normal derivative and the 0 for the second error term but I don't get where the -1 appeared from.

Reply

-  *AJ* says:
November 25, 2020 at 10:04 pm

It's because of the chain rule. In the original equation (1/2)(target – out_{o1})^2, when you end up taking the derivative of the (…)^2 part, you have to multiply that by the derivative of the inside. The derivative of the inside with respect to out_{o1} is 0 – 1= -1

Reply

17.  *Luqing* says:
October 28, 2020 at 11:46 pm

I read many explanations on back propagation, you are the best in explaining the process. Thank you very much.

Reply

18. Neural Networks – Feedforward Math – Shahzina Khan

19.  *apoola* says:
November 17, 2020 at 3:09 am

Matt, thanks a lot for the explanation….However, I noticed

net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775

should it be

net_{h1} = 0.15 * 0.05 + 0.25 * 0.1 + 0.35 * 1 = 0.3825

ie. 0.25 instead of 0.2 (based on the network weights) ?

Again I greatly appreciate all the explanation

Reply

-  *AJ* says:

November 25, 2020 at 10:21 pm

I think you may have misread the second diagram (to be fair its very confusingly labeled). Take a look at the first diagram in the section “The Backwards Pass.” Here we see that neuron o_1 has associated weights w5 & w6. If you look back at the first diagram, w5 & w6 are the top two labeled weights, so it would follow logically that neuron h1 is has the weights w1 & w2.
W2 has a value of .20, which is consistent with the way he performed the other calculations.

Reply

20. *gellaz* says:
November 24, 2020 at 11:04 am

Great explanation Matt! I really appreciate your work. I noticed a small mistake:
I noticed a small mistake at the end of the post:
net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1
should be:
net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1

Reply

21. طيب مظهر says:
November 25, 2020 at 2:09 pm

When calculating for w1, why are you doing it like :

Eo1/OUTh1 = Eo1/NETo1 * NETo1/OUTh1

and not like:

Eo1/OUTh1 = Eo1/OUTo1 * OUTo1/NETo1 * NETo1/OUTh1

Why are you going from Eo1 to NetO1 directly, when there is OUTo1 in the middle.

Reply

- *Aravinth M* says:
December 25, 2020 at 3:48 am

Eo1/OUTh1 = Eo1/OUTo1 * OUTo1/NETo1 * NETo1/OUTh1. Thanks for giving this explanation bro.

Reply

22. *Hemant* says:
December 21, 2020 at 10:41 am

any reason why back propagation is necessary ? Can we not do this with just forward propagation in a brute force way ? or is the forward propagation is somehow much slower than back propagation.

given that we have a network with weights,
can we not get dE(over all training examples)/dWi as follows:

[1] store current error Ec across all samples as sum of [Oactual – Odesired } ^2 for all output nodes of all samples
[2] simply change the Wi by say 0.001 and propagate the change through the network and get new error En over all training examples.
[3] then set Wi back to its old value. (so we do not mess it up for another Wi)
[4] and then dE/dWi = (En – Ec) / 0.001
[5] Do this for all weights to get all weight sensitivities.
[6] then change all weights Wi = Wi – dE/DWi * learning rate
[7] propagate through the network get Ec
[8] Repeat 1 to 7 until Ec is lower than acceptable error threshold

Reply

23.  *clhruv* says:
January 2, 2021 at 11:20 am

Great article! Just what I was looking for, thank you.

Reply

24.  *Adem* says:
January 2, 2021 at 4:36 pm

Thank you for your very well explained paper.

Reply

25.  *Elias* says:
January 6, 2021 at 2:09 pm

I think u got the index of w3 in neto1 wrong. It should be 2 or i am wrong ?

Reply

26.  *dajisafe* says:
January 6, 2021 at 5:20 pm

I noticed the exponential E^-x where x = 0.3775 (in sigmoid calculation) from my phone gives me -1.026 which is diff from math/torch.exp which gives 0.6856. Why is this different?

Reply

27. Gradient descent and loss landscape animations of neural networks in Python - Prog.world

28. ▷ Cum Se Creează O Rețea Neuronală în JavaScript în Doar 30 De Linii De Cod | Routech

29. *Minh Anh Trần* says:
January 27, 2021 at 11:40 pm

    Really great explaination, thank you very much !!! But is there any intuition why the value of derivative E_o1 to out_o1 is equal to the value of derivative E_total to out_o1 = .74136507 ?

    Reply

30. *prakash muniyappa* says:
February 2, 2021 at 4:16 am

    I tried building model along these lines.
    Innumerable thanks!! the results match exactly as specified.
    Matt, thanks a lot for making dummies like me understand this with very simple example.
    cheers!

    Reply

31. *Alaa* says:
February 14, 2021 at 7:39 pm

    Great article but how about more than one training data? thank you

    Reply

32. *Qingqing* says:
February 22, 2021 at 11:53 pm

    Hi Matt,
    Thanks for your post. I found it super useful!
    I followed your steps and built my NN model, fed some data into it.
    A weird thing happened.
    The loss does converge but the values of the whole prediction dataset are always greater than the target dataset (a significant difference exists between the mean of them). It seems that the prediction cannot reach the range of the target.

    Is there something wrong with my NN design? or the initialization?

    Thank you in advance

    Reply

33. *MariaLuisa* says:
February 23, 2021 at 10:02 am

    Love it!

Reply

34.  *Daniele* says:
March 19, 2021 at 7:34 am

Finally I understood backpropagation and its meaning!

Reply

-  *Daniele* says:
March 19, 2021 at 7:35 am

I am going to share this link on a forum of my uni course, if is ok!

Reply

35. Understanding difference between gradient of sigmoid vs softmax in the context of back propagation ~ Mathematics ~ mathubs.com

36.  *Upulie Han* says:
April 5, 2021 at 4:24 am

Took me some time to figure out that all the calculations have have been done only up to 9 decimal places.

Reply

37.  *Kuba* says:
April 5, 2021 at 7:26 am

You are a great teacher! <3 I wish You all kinds of good in Your life my friend!

Reply

38.  *Matt* says:
April 7, 2021 at 9:58 am

Just to be clear because I'm the second person to make this mistake. If you are getting 0.3825 instead of 0.3775, swap your w2 and w3, and also check your w6 and w7 to see if you made the same mistake there. w2 leads to the first hidden node NOT the second one. (The diagram could also be more clearly labeled.)

Reply

39.  *Cyril* says:
April 12, 2021 at 10:03 am

What about providing several different input values to the net? My problem is that it does learn to provide the expected values, but it only works with one training example. If I

repeatedly give it different examples with corresponding targets, it only memorizes the last one and outputs the last target no matter what values I give to it. I tried changing weights by a value which is an average of weight changing values for each of the examples – did not work, the training process got stuck with outputs holding averages of each target vector (so if we have target vectors (1, 1), (0, 0) and (0, 1), the outputs are (0.33, 0.67)), and it also fails to minimize the cost which is an average of costs for each training example. How do we manage network training with a bunch of different input and target vectors?

Reply

-  *Cyril* says:
  April 12, 2021 at 10:10 am

  I stand corrected, "it only memorizes the last one and outputs the last target no matter what values I give to i" it outputs values in accordance with the weights adjusted to output the last target vector when the net is given the last input vector from the example set

  Reply

40.  *Cyril* says:
April 12, 2021 at 10:14 am

What about providing several different input values to the net? My problem is that it does learn to provide the expected values, but it only works with one training example. If I repeatedly give it different examples with corresponding targets, it only memorizes the last one and outputs values in accordance with the weights adjusted to provide the last target vector when given the last example from the set. I tried changing weights by a value which is an average of weight changing values for each of the examples – did not work, the training process got stuck with outputs holding averages of each target vector (so if we have target vectors (1, 1), (0, 0) and (0, 1), the outputs are (0.33, 0.67)), and it also fails to minimize the cost which is an average of costs for each training example. How do we manage network training with a bunch of different input and target vectors?

Reply

41.  *Cyril* says:
April 12, 2021 at 10:33 am

why my comments get deleted?

Reply

-  *Mazur* says:
  April 18, 2021 at 5:20 pm

  They're not – I don't delete any. Not sure what's up.

  Reply

42.  *Zewd* says:
    April 17, 2021 at 8:21 am

    Thank you! it so helpful, but I couldn't find the true value to update weight1. Could you help me?

    Reply

43. 58 Resources To Help Get Started With Deep Learning (In TF) – 365 Data Science

44. 58 Resources To Help Get Started With Deep Learning (In TF) - Ready AI.M

45.  *Anwaar Gharaibeh* says:
    April 26, 2021 at 11:50 am

    the claculation of nethi is wrong
    0.075+0.020+0.350= 0.445

    Reply

46.  *Leon Qi* says:
    May 21, 2021 at 3:23 pm

    Nice explanation! I got a question: what if we try to train two neural networks, one is wide and the other is deep, with the same number of neurons and training epochs. Which one would be faster?

    Reply

    -  *Mehr* says:
      June 4, 2021 at 9:03 am

      It shouldn't be different in time for train or run but it can be different in performance. You can find articles to give you an idea of how many layers or neurons per layer is better for this or that type of problem. But it isn't a clear fix answer for that and you can find the best performance with try and error.

      Reply

47. Let's build your 1st Artificial Neural Network Model using Python (Part 2) – Ramsey Elbasheer | History & ML

Comment navigation

← Older Comments

Leave a Reply

Enter your comment here

Subscribe for Updates

Enter your email address to follow this blog and receive notifications of new posts by email.

Join 919 other followers

Email Address:

Enter your email address

Follow

About

Hey there! I'm the founder of Preceden, a web-based timeline maker, and data wrangler at Help Scout, a company that makes customer support tools. I also built Lean Domain Search and many other software products over the years.

Search Search … Search

Follow me on Twitter

My Tweets

Blog at WordPress.com.

%d bloggers like this:

:)