



Matt Mazur

[Home](#)
[About](#)
[Archives](#)
[Contact](#)
[Projects](#)

Follow via Email

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

Join 2,842 other followers

About

Hey there! I'm the founder of [Preceden](#), a web-based timeline maker, and a data analyst consultant. I also built [Lean Domain Search](#) and [many other software products](#) over the years.



Follow me on Twitter

[My Tweets](#)

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.

If this kind of thing interests you, you should [sign up for my newsletter](#) where I post about AI-related projects that I'm working on.

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](#).

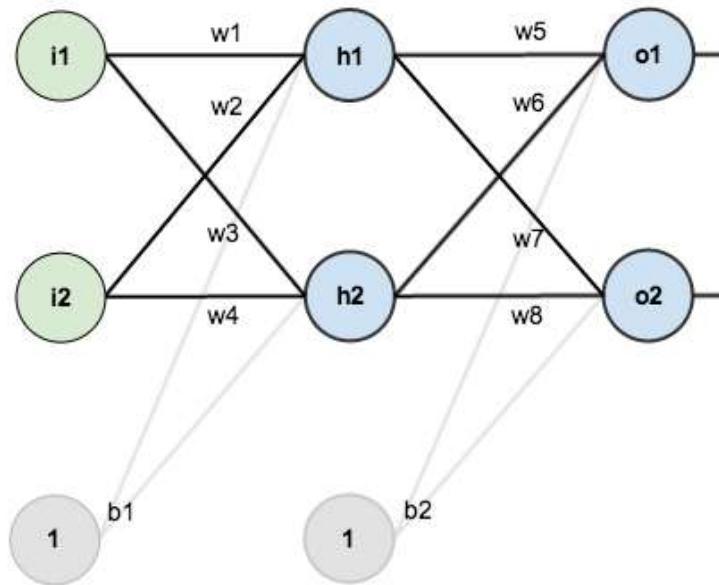
Additional Resources

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.

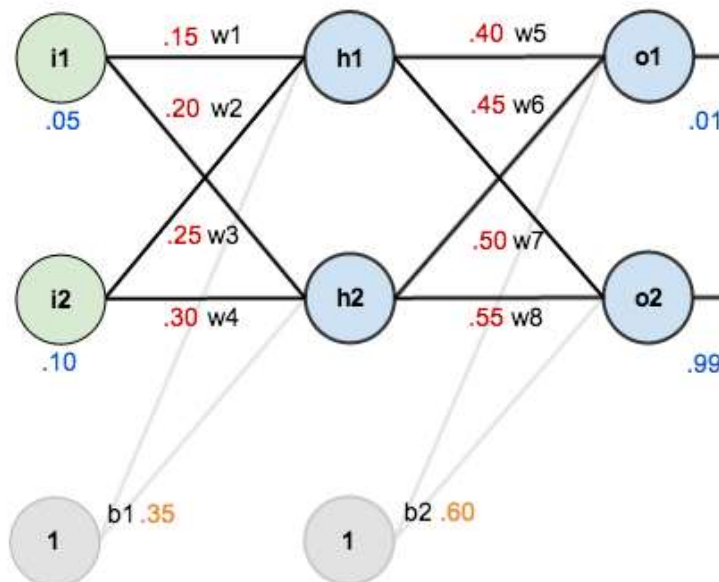
Overview

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, let's 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 through 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:

$$E_{o1} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

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

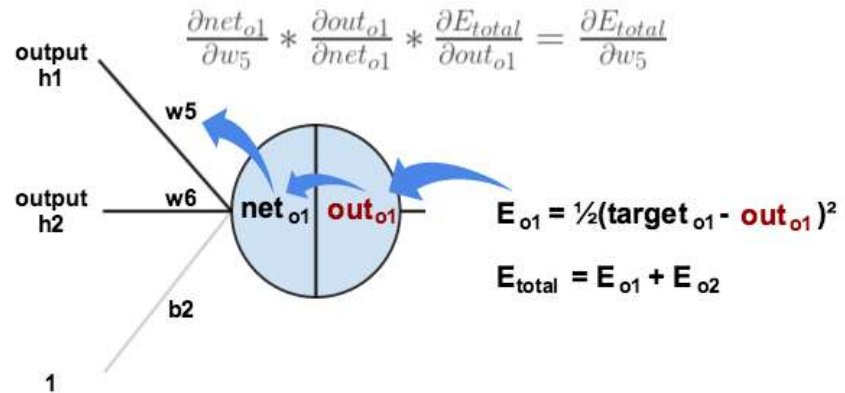
Consider w_5 . We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

$\frac{\partial E_{total}}{\partial w_5}$ 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:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \text{out}_{o1}} * \frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} * \frac{\partial \text{net}_{o1}}{\partial w_5}$$

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?

$$E_{\text{total}} = \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^2 + \frac{1}{2}(\text{target}_{o2} - \text{out}_{o2})^2$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = 2 * \frac{1}{2}(\text{target}_{o1} - \text{out}_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{\text{total}}}{\partial \text{out}_{o1}} = -(\text{target}_{o1} - \text{out}_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$-(\text{target} - \text{out})$ is sometimes expressed as $\text{out} - \text{target}$

When we take the partial derivative of the total error with respect to out_{o1} , the quantity $\frac{1}{2}(\text{target}_{o2} - \text{out}_{o2})^2$ 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:

$$\text{out}_{o1} = \frac{1}{1 + e^{-\text{net}_{o1}}}$$

$$\frac{\partial \text{out}_{o1}}{\partial \text{net}_{o1}} = \text{out}_{o1}(1 - \text{out}_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Finally, how much does the total net input of o_1 change with respect to w_5 ?

$$\text{net}_{o1} = w_5 * \text{out}_{h1} + w_6 * \text{out}_{h2} + b_2 * 1$$

$$\frac{\partial \text{net}_{o1}}{\partial w_5} = 1 * \text{out}_{h1} * w_5^{(1-1)} + 0 + 0 = \text{out}_{h1} = 0.593269992$$

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 \$\eta\$](#) (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).

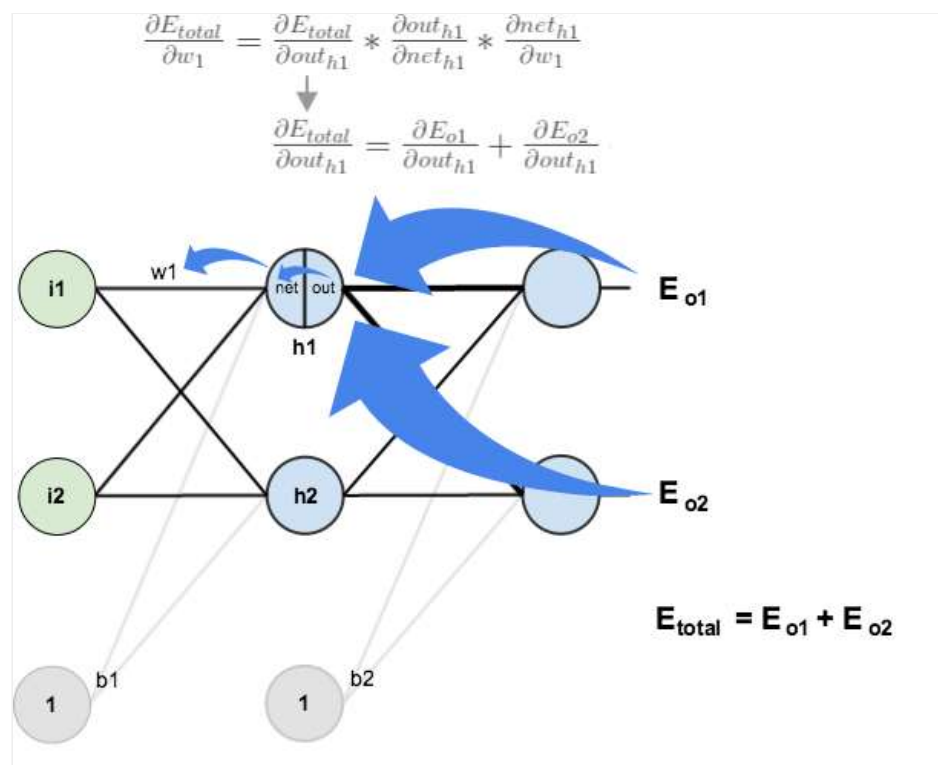
Hidden Layer

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:



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 $\frac{\partial net_{h1}}{\partial w_1}$ 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:

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

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

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

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:



113 bloggers like this.

Related

Experimenting with a
Neural Network-based
Poker Bot
In "Poker Bot"

The State of Emergent
Mind
In "Emergent Mind"

I'm going to write more
often. For real this time.
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](#).

[← Introducing
ABTestCalculator.com, an Open
Source A/B Test
Significance Calculator](#)

[TetriNET Bot Source Code Published
on Github →](#)

798 thoughts on “A Step by Step Backpropagation Example”

[← Older Comments](#)



chantiq1000x

— June 21, 2018 at 9:44 am

can you tell me why $h1 = w1*x1 + w2*x2 + b$? why not $h1 = w1 * x1 + w3* x3 + b$

[Reply](#)



Joseph

— August 1, 2018 at 9:28 pm

The labeling is slightly ambiguous because the weights were not put right on top of the links/edges. $w2$ is not going from $i1$ to $h2$, it is going from $i2$ to $h1$. That is, the weights are labeled to the left of their respective links/edges, not to the right. This is a little bit more obvious in the first graph.

[Reply](#)

George Skrimpas

— June 21, 2018 at 12:23 pm

I think the correct formula for $H1$ is:

$netH1 = w1*i1 + w3*i2 + b1*1$

i use $w3$ instead of $w2$, since $I2$ is linked to $H1$ via $w3$.

Will you plz confirm?

[Reply](#)



deepak

— June 27, 2018 at 11:31 am

many errors dude mainly the computation of $d(h1)/d(o2) = -(0.01 - 0.772928465) * (0.772928465 * (1 - 0.772928465)) * 0.45$

[Reply](#)

**Suman**

— June 30, 2018 at 4:06 pm

It really helped me to understand how back propagation works. Keep up the good work.

[Reply](#)**mcavidya**

— July 3, 2018 at 9:11 pm

Thanks Mazur for this numerical example. Such an example provides the best way to learn the working of an algorithm. It helped me a lot. Many thanks.

Going through the example, I was wondering whether the biases b_1 , b_2 are being treated as constants. Is it not necessary to adjust their values also?

[Reply](#)**Mayank Gupta**

— August 6, 2018 at 5:10 am

Yes. In this article, biases were treated as constants to keep things simple. If you understood the math explained in this article, you can easily update the biases as well. In reality, biases are also updated.

[Reply](#)**enginbozaba**

— September 11, 2018 at 1:24 pm

Are we updating the bias values by setting up the dE / db_1 and dE / db_2 chains?

[Reply](#)**Pony**

— July 3, 2018 at 10:16 pm

This is great! Thank you for the step by step explanation

[Reply](#)**Jerome lemoine**

— July 6, 2018 at 3:04 pm

Hi matt

Great document, very pedagogic !

Maybe a little mistake in the calculation of the net_{h1} and net_{h2} :

A Step by Step Backpropagation Example – Matt Mazur

$$\text{net_h1} = w1 \times i1 + w3 \times i3 + b1$$

(Analog mistake for net_h2)

The mistake is communicated to the numerical application also

[Reply](#)**lamductan**

— July 25, 2018 at 4:56 am

This explanation and visualization is very well understanding. It helps me so much because it takes me a lot of time to know how really backpropagation does. Thank you very much.

[Reply](#)

ping!

[A Step by Step Backpropagation Example – Deep Learning](#)**jeffminich**

— July 31, 2018 at 5:21 pm

Outstanding explainer for back propagation. Thanks, Matt!

[Reply](#)**Mayank Gupta**

— August 6, 2018 at 5:07 am

I was a bit frustrated with this backpropagation topic and was struggling to have a clear mental picture of backpropagation.

Your article radically improved my understanding. Your article was so clear that I was actually able to write my own code to implement backpropagation. Thank you very much.

If you ever plan to expand on this article, I request you to add some details about how weights are updated for all the samples (your article explained the case for one sample.)

Thanks again for this wonderful article!

[Reply](#)**arnulfo**

— August 6, 2018 at 11:07 pm

Reblogged this on [conlatio](#).

[Reply](#)

**Amin**

— August 7, 2018 at 9:51 am

Goog job! But this is 3-layer network only. If it is 4-layer then how we calculate the $dE_{total}/dout(h1)$?

$dE_{h2}/dout(h1)$ will not be know. Because we don't have the value for dE_{h2} (the error for hidden layer 2). I need an explanation here. Thanks!

[Reply](#)
**Zhijie Chen**

— August 15, 2018 at 11:12 am

You use the value that comes from the previous layer. For instance $dE/dout\ i1 = dE/dnet\ h1 * dnet\ h1/dout\ i1 = dE/dout\ h1 * dout\ h1 / dnet\ h1 * dnet\ h1/dout\ i1$, in which $dE/dout\ h1$ and $dout\ h1 / dnet\ h1$ have been calculated by the previous layer.

[Reply](#)
**Knut Leiß**

— November 14, 2018 at 4:36 am

I got the same problem and I don't understand the answer.

Why is $dE/dout\ i1 = dE/dnet\ h1 * dnet\ h1/dout\ i1$?

This ignores the contribution of out i1 to the error through h2!?

So, I assume the term $dE/dnet\ h2 * dnet\ h2/dout\ i1$ needs to be added!?

But then, why is it OK to just use the sum, mathematically?

[Reply](#)
**Frank**

— August 15, 2018 at 3:27 pm

Question: do all neurons in a layer use the same bias weight or is there a individual weight per neuron? I.e. In the example it looks like that $b1$ is used for both hidden neurons and $b2$ for both output neurons.

Btw: nice and easy to follow example!

[Reply](#)
**Aaron**

— October 23, 2018 at 1:20 am

In reality each bias would be adjusted too, but in the example seems like they are only focusing on weights, and specifically the hidden layers.

[Reply](#)

ping! [Implementing a flexible neural network with backpropagation from scratch](#)



anil

— August 21, 2018 at 12:03 am

i think this is best explain backpropagation with detail kudos!

[Reply](#)

ping! [Important Links | Tejalal Choudhary](#)



Garrett

— August 24, 2018 at 11:07 pm

This was a fantastic write up. I am using it to study the algorithm while currently in an AI course at uni. I was wondering if you could add to this and describe momentum in the same way. Thank you

[Reply](#)



Rafay

— August 27, 2018 at 1:09 pm

Thanks a lot Matt for making this. Your blog and the Stanford's CS231n lectures are the best resources on this.

[Reply](#)

ping! [Deep learning for product managers – part 1 – Kai's notebook](#)



Alan Wake

— August 31, 2018 at 7:52 am

Thank you very much! I've been looking exactly for this

[Reply](#)

ping! [【每日AI收集】BP神经网络 - 每日AI](#)



Sneha

— September 9, 2018 at 5:07 am

This page is very helpful to solve the problems.

Amazing page !!

Thank you !!

[Reply](#)

**Kevin**

— September 15, 2018 at 1:22 am

Thanks for the great example. One question though:
During the backpropagation, you find the new values for w_5 - w_8 and continue backpropagating the previous hidden layer. There you need the value of $dnet_o1/dout_h1$, which equals w_5 . Do we use the original value of w_5 , 0.4, as it is done here or its updated value, 0.35891648 ?

[Reply](#)**Artem**

— September 15, 2018 at 12:20 pm

Thank you for explanation. I didn't get one thing. Why we multiply by -1 derivative of total error? is this because target is constant and out1 is function?

[Reply](#)**Abin Singh**

— September 21, 2018 at 4:08 am

You said in the beginning that we are going to work with only a single training set (1 row with 2 features). Then how we are getting two outputs. It should be one right.

Can you please clarify?

[Reply](#)**Jimmy Johnson**

— September 25, 2018 at 1:00 am

Very nice explanation.
Thanks for taking the time to do this.

[Reply](#)**swapna**

— September 27, 2018 at 9:36 am

its really very helpful..thanks for such a wonderful doc

[Reply](#)

**ynot werk**

— September 29, 2018 at 3:14 pm

Carrying out the same process for h_2 we get:

$out_{[h2]} = 0.596884378$

I got 0.666522 for h_2 because i did: $activation_function(0.25 * 0.05 + 0.3 * 0.1 + 0.65 * 1)$

[Reply](#)

**R S SARATHI KANNAN**

— October 5, 2018 at 11:52 pm

Hi,

It is a good work. I am struck at here, Following the same process for $\frac{\partial E_{o2}}{\partial out_{[h1]}}$, we get:

$\frac{\partial E_{o2}}{\partial out_{[h1]}} = -0.019049119$ (partial derivative of $H1_{out}$ from $E2$)...
how to calculate out_{H1} from $E2$

[Reply](#)

**Martin Ennemoser**

— October 13, 2018 at 4:10 am

Thanks for this awesome explanation.

[Reply](#)

**Kartikey arora**

— October 18, 2018 at 10:54 am

This article was really very helpful for me as i was very confused and stuck with the backprop calculations and u just explained it with an example in a very nice way thanks a lot

[Reply](#)

Ping!

[A simple natural language category classifier with Keras – GrownSoftware](#)

**Ravi**

— October 21, 2018 at 3:51 am

Final weights after 10,000 iterations are

0.378509988350132

0.657019976700263

0.477300705684786

0.754601411369569
-3.892977198865724
-3.868416687703616
2.861810948620059
2.925688966771616

[Reply](#)



Ana

— October 30, 2018 at 2:23 pm

I've got the same, is that right? aren't weights supposed to be between 1-10?

[Reply](#)

ping!

[Neuronal networking in VBA+Excel](#)



Aaron

— October 22, 2018 at 11:01 pm

This is an awesome post, I was confused about how the error propagates through the hidden layer. Working through each step made the equations seem intuitive. Someone should really make an animation for this.

[Reply](#)

ping!

[\[D\] Is there a book or resource about machine learning that teaches different algorithms with step by step calculations for simple examples? Like how a mathematics or physics book would. | Premium Blog! | Development code, Android, ios anh Tranning.IT](#)



Raja

— October 24, 2018 at 7:16 pm

Hi Experts, I tried this approach, for some reason my weights and error keeps increasing, doesn't seem to reduce for certain inputs, any idea whats happening ?

[Reply](#)



eduhelp2016

— October 27, 2018 at 5:47 am

Lovely!!!!!!!!!!!!

[Reply](#)



T.smart

— October 29, 2018 at 3:08 am

it is my answer,
0 0.291027774
1 0.283547133
2 0.275943289
3 0.268232761
4 0.260434393
5 0.252569176
6 0.244659999
7 0.236731316
8 0.228808741
9 0.220918592
10 0.213087389
11 0.205341328
12 0.197705769
13 0.190204742
14 0.182860503
15 0.175693166
16 0.168720403
17 0.16195725
18 0.155415989
19 0.149106135
20 0.14303449
21 0.137205276
22 0.131620316
23 0.126279262
24 0.121179847
25 0.116318143
26 0.111688831
27 0.107285459
28 0.103100677
29 0.099126464
30 0.095354325
31 0.091775464
32 0.088380932
33 0.085161757
34 0.082109043
I am not sure that it is right.

[Reply](#)



PARAMESHA K

— October 31, 2018 at 9:05 pm

Whether the BP should run for all the training samples or individual samples?

[Reply](#)

ping!

[Understanding Evolved Policy Gradients – AI+ NEWS](#)



purnenduprabhat

— November 11, 2018 at 3:38 am

Excellent explanation! Cleared my doubt.

[Reply](#)**Sebastian**

— November 11, 2018 at 9:45 am

Great explanation. Thanks. Many authors who write books consider themselves as being smart just because they provide some complicated formulas from wikipedia with 10,000 different index symbols. But I do not think they are smart at all. They do not know their audience. This simple example is better than all books that I have which try to explain back propagation with fomrulas... I did not realize it is that simple. It is just taking derivative over and over again. Thank you very much!

[Reply](#)**Santiago Fernando Gomez**

— November 19, 2018 at 5:22 pm

This is amazing, the best explanation I have found over the internet. Great job!

[Reply](#)[← Older Comments](#)

Leave a Reply

[Blog at WordPress.com.](#)