Matt Mazur

A Step by Step Backpropagation Example

March 17, 2015February 23, 2024 / Mazur

Background

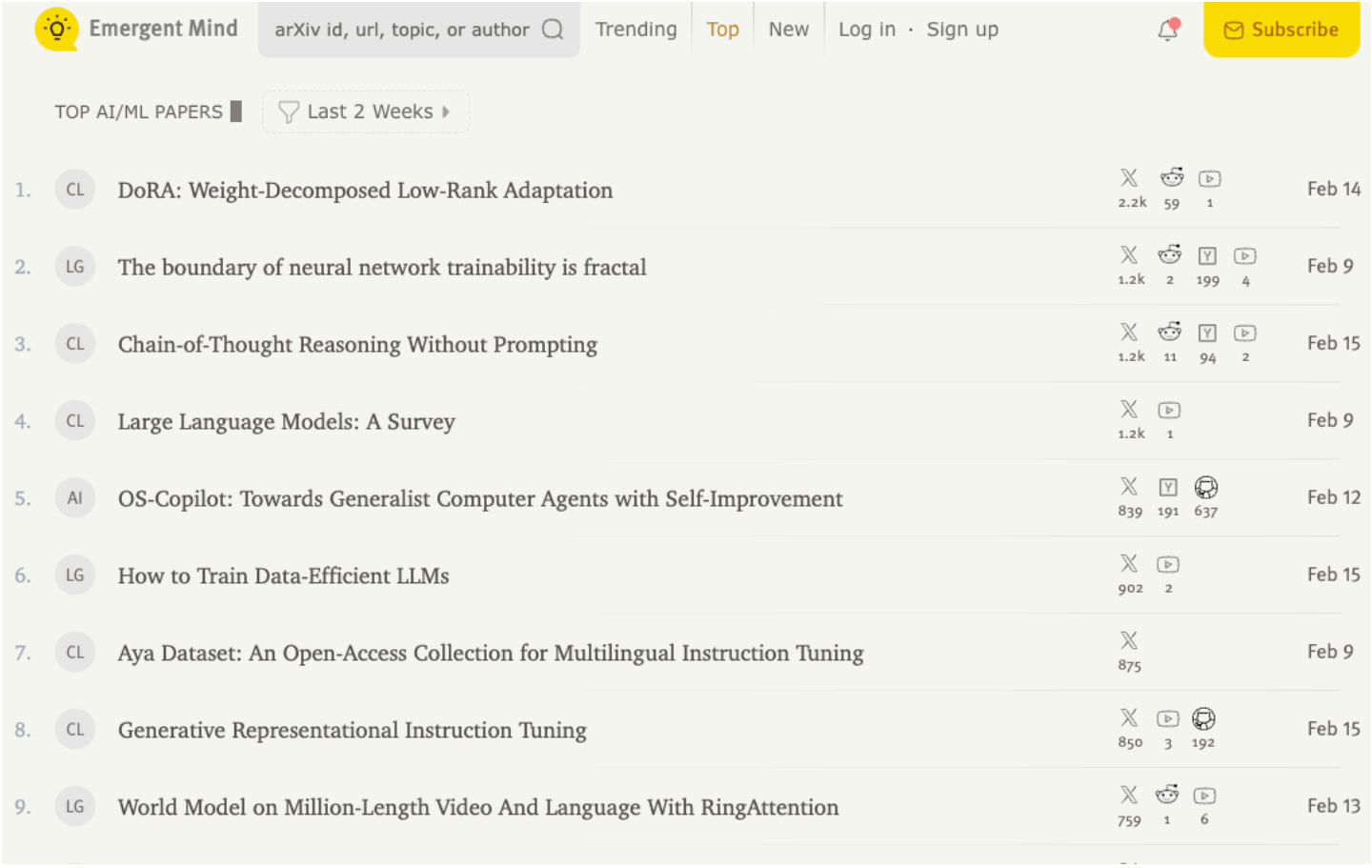
Backpropagation is a common method for training a neural network. There is **no shortage of papers (https://www.google.com/search?q=backpropagation+algorithm)** online that attempt to explain how backpropagation works, but few that include an example with actual numbers. This post is my attempt to explain 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 (https://github.com/mattm/simple-neural-network)**.

Continue learning with Emergent Mind

If you find this tutorial useful and want to continue learning about AI/ML, **I encourage you to check out Emergent Mind, (https://www.emergentmind.com/)** a new website I'm working on that uses GPT-4 to surface and explain cutting-edge AI/ML papers:



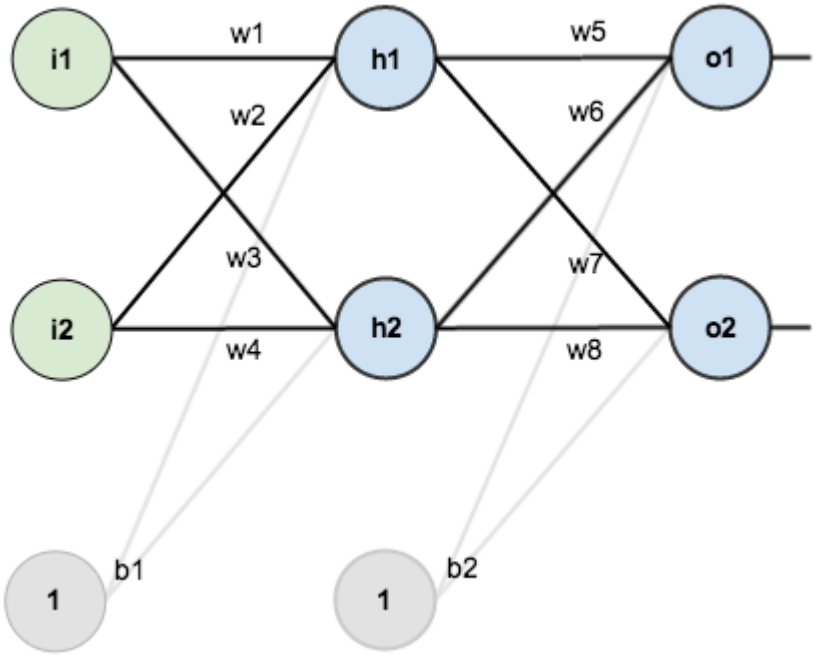
(https://mattmazur.com/wp-content/uploads/2024/02/xnapper-2024-02-23-14.38.15.png)
In time, I hope to use AI to explain complex AI/ML topics on Emergent Mind in a style similar to what you'll find in the tutorial below.

Now, on with the backpropagation tutorial…

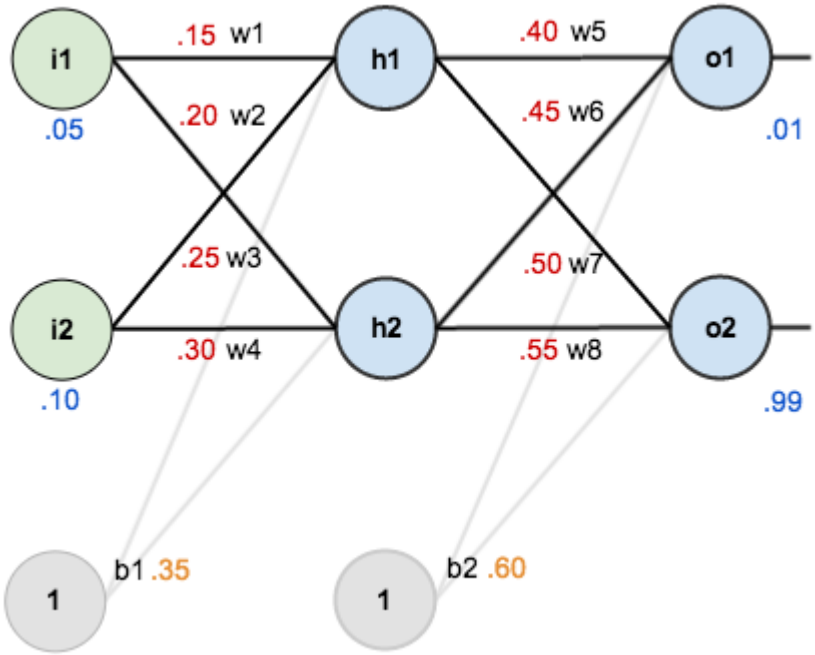
Overview

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 (http://web.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf)**.
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 (http://en.wikipedia.org/wiki/Backpropagation#Derivation)** and sum them to get the total error:

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

Some sources (http://www.amazon.com/Introduction-Math-Neural-Networks-Heaton-ebook/dp/B00845UQL6/ref=sr_1_1?ie=UTF8&qid=1426296804&sr=8-1&keywords=neural+network) 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 (http://en.wikipedia.org/wiki/Backpropagation#Derivation)**].
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

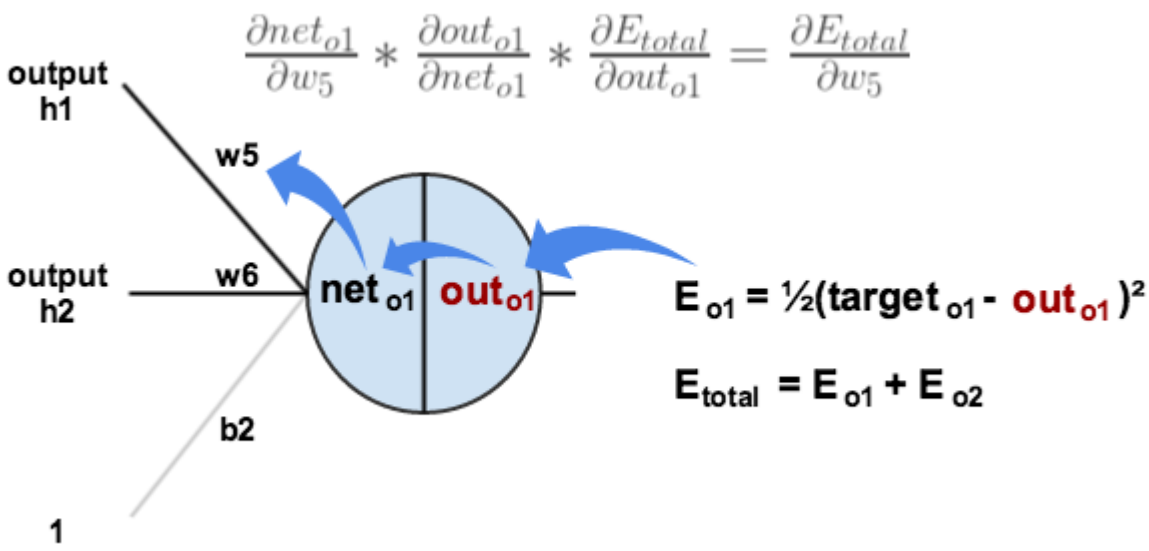
Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

$\frac{\partial E_{total}}{\partial w_5}$ 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 (http://en.wikipedia.org/wiki/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 (http://en.wikipedia.org/wiki/Logistic_function#Derivative)** 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 (http://en.wikipedia.org/wiki/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 (http://en.wikipedia.org/wiki/Delta_rule) sources (http://aima.cs.berkeley.edu/) use α (alpha) to represent the learning rate, **others use (https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf)** η (eta), and **others (http://web.cs.swarthmore.edu/~meeden/cs81/s10/BackPropDeriv.pdf)** 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:

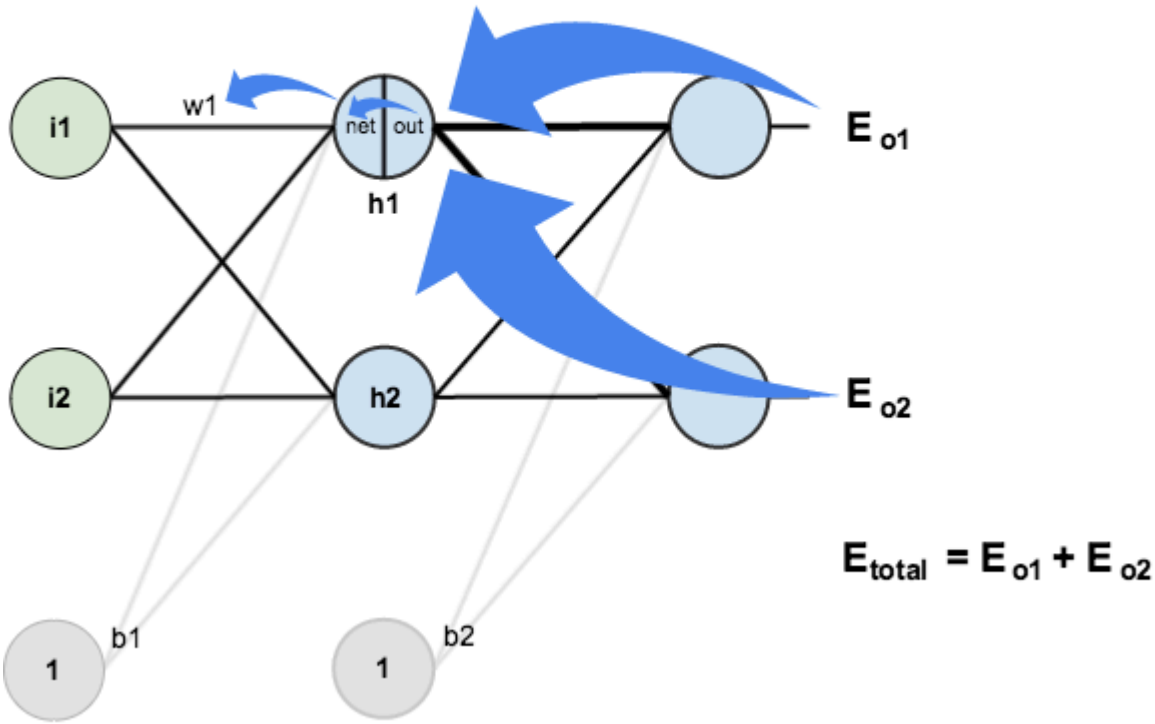
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Visually:

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

$$\downarrow$$

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



(https://mattmazur.com/wp-content/uploads/2015/03/nn-calculation.png)

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 (https://mattmazur.com/contact/)**. Thanks!

And while I have you…

Again, if you liked this tutorial, please check out **Emergent Mind (https://www.emergentmind.com)**, a site I'm building with an end goal of explaining AI/ML concepts in a similar style as this post. Feedback very much welcome!
Categories: Machine Learning Tags: ai, backpropagation, machine learning, neural networks

1,093 thoughts on "A Step by Step Backpropagation Example"

1. **wth** says:
   SEPTEMBER 2, 2023 AT 1:12 PM
   Why are you calculating net_h1 with w1 and w2 instead of w1 and w3, which are the weights leading to h1?

   REPLY
   1. **Ehlert** says:
      SEPTEMBER 4, 2023 AT 1:03 AM

You must take a closer look! net_h1 is connected with w1 and w3!
w2 and w3 are the names of the lines left, not for the lines right ;-)

REPLY

1. **Besfort Ramadani** says:
   JANUARY 7, 2024 AT 2:15 AM
   You are correct. But, in the calculation of the net input h1, he uses the value of the weight w2 which is 0.2. The value for w3 is 0.25 and therefore the calculation of the net input of h1 should be:
   h1 = 0.15 * 0.05 + 0.25 * 0.1 + 0.35

   I hope it is correct.

   REPLY
2. **yangao** says:
   SEPTEMBER 6, 2023 AT 3:28 AM
   Thanks matt, this blog help me learn a lot

   REPLY
3. **ion** says:
   SEPTEMBER 10, 2023 AT 3:02 AM
   excellent!!

   REPLY
4. **Bindeshwar Kushwaha** says:
   SEPTEMBER 26, 2023 AT 10:05 PM
   Very intuitive and excellent.

   REPLY
5. **Pavel** says:
   SEPTEMBER 29, 2023 AT 5:32 PM
   There is a wrong value for ‘target o1’ when calculating ‘E o1.’ You are using 0.01 instead of 0.1, according to this picture https://matthewmazur.files.wordpress.com/2018/03/neural_network-9.png

   REPLY
6. **basil Ahmad** says:
   NOVEMBER 1, 2023 AT 3:31 AM
   thanks for the super great explaination

   REPLY
7. **Yuriy** says:
   NOVEMBER 5, 2023 AT 8:05 AM
   What to do with the bias ? Or will it have an initial random value forever ?

   REPLY
8. **diamonds** says:
   NOVEMBER 6, 2023 AT 2:10 PM
   The best explanation!

   REPLY
9. **Rabea** says:
   NOVEMBER 8, 2023 AT 5:23 PM
   This is great

REPLY

10. **Eesaan** says:
NOVEMBER 12, 2023 AT 4:41 AM
Thanks a ton for this!

REPLY

11. **Eesaan** says:
NOVEMBER 12, 2023 AT 4:42 AM
Thank you very much sir!

REPLY

12. **MC** says:
NOVEMBER 12, 2023 AT 3:45 PM
VThank you

REPLY

13. **Homayoon Ranjbar** says:
NOVEMBER 24, 2023 AT 2:55 PM
Excellent post.
Many people evade the dirty calculations of back-propagation and tend to just mention it verbally and not with writing equations like you did, that being said, well done!

REPLY

14. **Flexe** says:
DECEMBER 4, 2023 AT 8:16 PM
Hey,
That's a very nice blog. How can we adapt the code to use mini-batch insteed of online learning? I have some problems with adaptations. Can someone help me out ?

Thanks

REPLY

15. **Morgan** says:
DECEMBER 5, 2023 AT 9:49 AM
Shouldn't the dE02/dOutH1 =-0.019049119 actually be = -0,0171442064749798057954250 8125??
Regards Morgan

REPLY

16. **Flexe** says:
DECEMBER 5, 2023 AT 11:03 AM
Hallo guys,
Thanks you Matt for this great job.
can someone help me out to adapt this code so that it uses mini batch insteed of online learning ?
Thanks.

REPLY

17. **AP51** says:
DECEMBER 13, 2023 AT 1:44 PM
Great demo !
I wanted to check any details so i tried to implement it on Google Colab with tensorflow and Keras and i checked any line , any detail … i get the sames values for these 2 backpropagations on

all parameters , weights , gradients , losses … check my link : check my link
https://colab.research.google.com/drive/13i8YefcJfOo4Sg6QseER-AYwBUKQDl15?usp=sharing
Of course it can be presented a better way
REPLY

18. **Deribew Shimelis** says:
JANUARY 26, 2024 AT 1:59 AM
best one

REPLY

19. **Andres Namm** says:
JANUARY 29, 2024 AT 1:44 AM
This post was useful in 2017 and is the best post about NNs in 2024. Truly world changing!

REPLY

20. **Shahab Ansari** says:
JANUARY 29, 2024 AT 6:42 AM
Great work indeed!!!

REPLY

21. **Bilal Khan** says:
FEBRUARY 3, 2024 AT 12:24 PM
Good Work, we must appreciate this effort.

REPLY

22. **ericguedespinto** says:
FEBRUARY 20, 2024 AT 5:42 AM
Very good explanation. This blog was the go to place when I was implementing a back propagation function in dart.
One suggestion I have is to abstract the derivative of the activation function and of the cost function, maybe with examples of alternative functions (ReLU, Huber loss, …) so that the modular feeling of training a NN is highlighted.

REPLY

23. **Paul** says:
APRIL 10, 2024 AT 1:18 PM
There's a misleading mistake at the beginning. net_h1 calculation should use w1 and w3 but you use w1 and w2 so all the calculations are wrong.

REPLY

1. **Arzel Patrick** says:
APRIL 23, 2024 AT 11:53 AM
Hy Paul

The calculation is as follow :

net H1 = w1 * i1 +w2 * i2 = 0.3375

w1= 0.15 , w2 = 0.20

i1 = 0.05 , i2 = 0.10 , b1 = 0.35

I checked all in a Python notebook where each step is checked and validated with the formulas. All formulas are correct.

REPLY

2. **Arzel Patrick** says:
   APRIL 23, 2024 AT 12:03 PM
   net H1 = w1 * i1 +w2 * i2 = 0.3375

   REPLY

   1. **Arzel Patrick** says:
      APRIL 23, 2024 AT 12:04 PM
      net H1 = w1 * i1 +w2 * i2 + b1 = 0.3375

      REPLY

24. **Ak daurah** says:
    APRIL 17, 2024 AT 10:39 AM
    Thank you 😊

    REPLY

25. **Nada** says:
    JULY 3, 2024 AT 4:45 AM
    i need same example but computing error with cross entropy..do you have it?

    REPLY

26. **Madhu Soman** says:
    AUGUST 22, 2024 AT 6:26 AM
    Great example. Searched at least 20-25 blogs for a step by step example before seeing your blog.

    REPLY

27. **Dan** says:
    AUGUST 27, 2024 AT 11:42 PM
    Matt isn't calculating gradients for the Biases. So his answer is correct is you don't touch the Biases. My code calculates gradients for biases and adjusts the biases during backprop, took me hours of debugging to figure out my code is correct …

    REPLY