Matt Mazur

- Home
- About
- Archives
- Contact
- Projects

Follow via Email

Click to follow this blog and receive notifications of new posts by email.

Join 2,785 other followers

Follow

About

Hey there! I'm the founder of Preceden, a web-based timeline maker. I also built Lean Domain Search and many other software products over the years.

Search …

Follow me on Twitter

Tweets by @mhmazı

Matt Mazur
@mhmazur

Replying to @mhmazur

It's worth checking out the URL this XSS-test points to: nerveux.xss.ht - not often you see so much vanilla JavaScript these days

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.

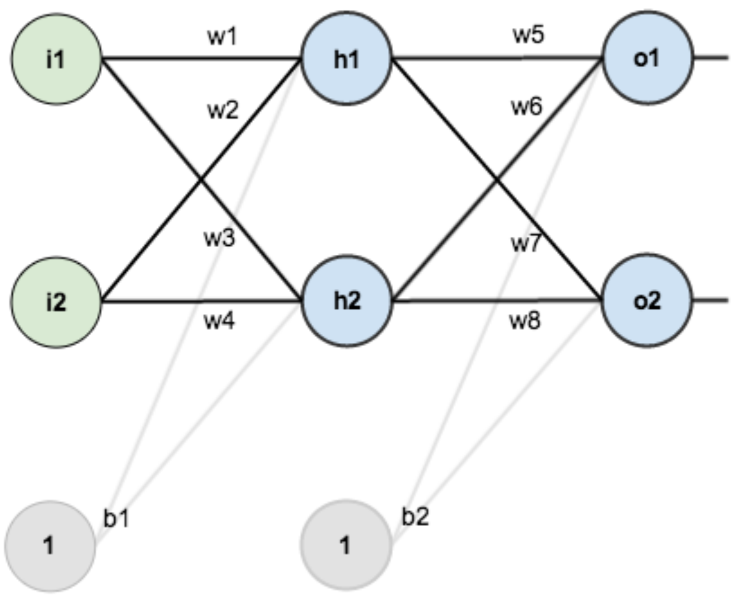
Additional Resources

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.

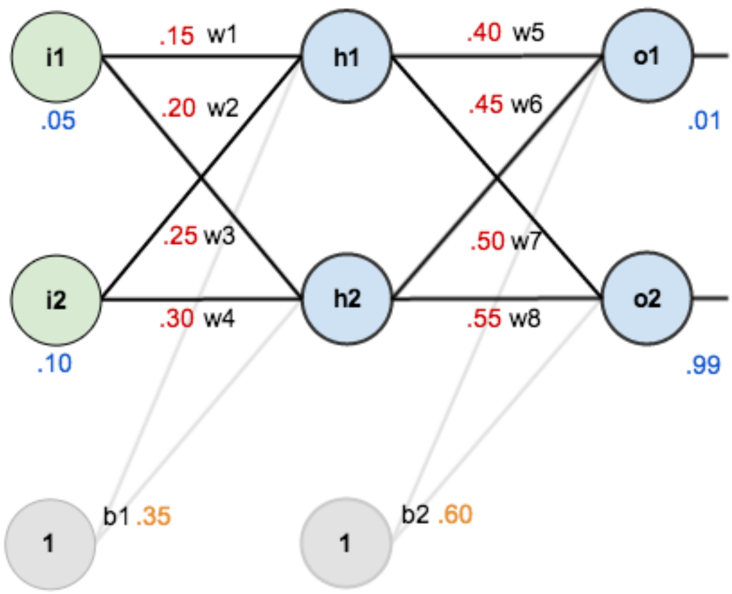
Overview

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

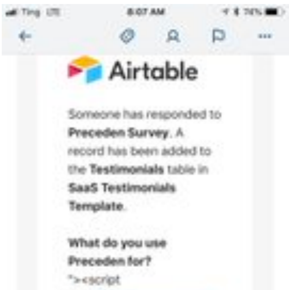
To begin, lets see what the neural network currently predicts given the weights

JavaScript these days

1h

Matt Mazur
@mhmazur

Something about these Preceden survey responses seems fishy to me...



2h

Matt Mazur
@mhmazur

A New Adventure: I'm Taking the Leap to Focus on Preceden and Analytics Consulting mattmazur.com/2018/09/04/a-n…

Sep 4, 2018

Matt Mazur Retweeted

Dave Martin
@itsdavemartin

Excited to share that I'm about 90% of the way to a beta release for tryhowdy.com 🙌

If you're a designer and you'd like a free beta account, just head to the URL above and enter your email.

Here's a quick preview of

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

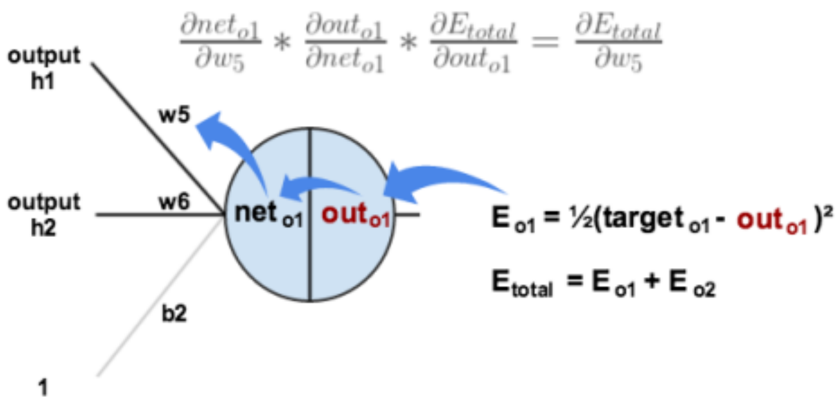
Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

> $\frac{\partial E_{total}}{\partial w_5}$ 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 re-write 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).

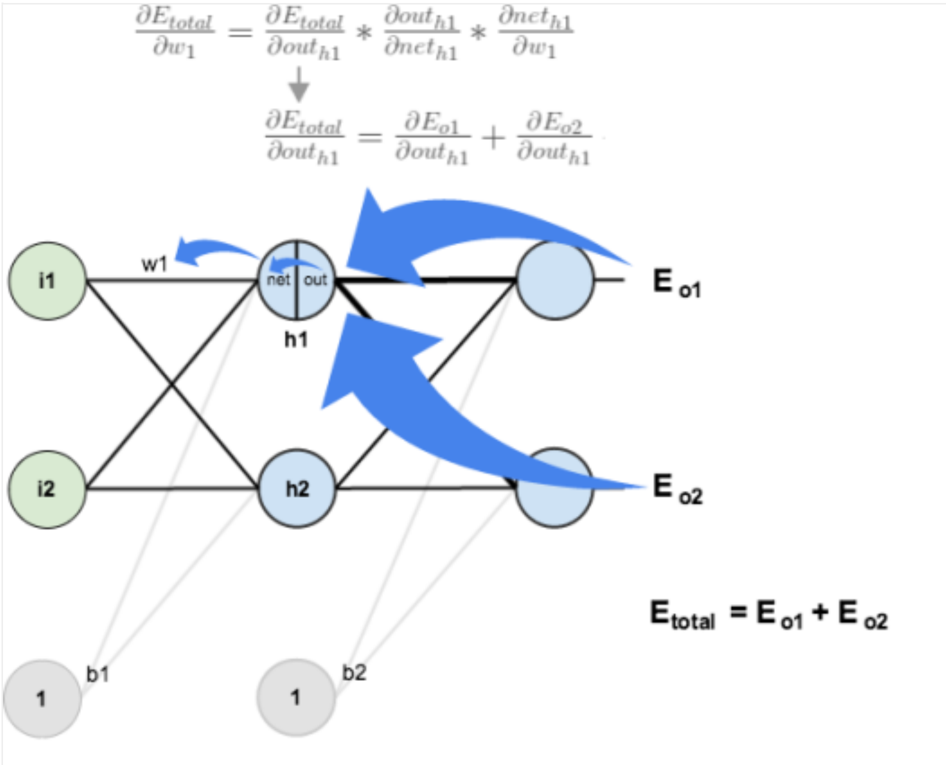
Hidden Layer

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:



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!

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 →

771 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 b1, b2 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

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:
net_h1= w1×i1 + w3×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 explaination 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 dEtotal/dout(h1)?

dEh2/dout(h1) will not be know. Because we don't have the value for dEh2 (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

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

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

← Older Comments

Leave a Reply

Enter your comment here...

Blog at WordPress.com.