Cross Validated

Backpropagation with Softmax / Cross Entropy

Asked 3 years, 2 months ago Active 1 month ago Viewed 78k times

40

43

I'm trying to understand how backpropagation works for a softmax/cross-entropy output layer.

The cross entropy error function is

$$E(t, o) = -\sum_j t_j \log o_j$$

with t and o as the target and output at neuron j, respectively. The sum is over each neuron in the output layer. o_j itself is the result of the softmax function:

$$o_j = softmax(z_j) = \frac{e^{z_j}}{\sum_j e^{z_j}}$$

Again, the sum is over each neuron in the output layer and z_j is the input to neuron j:

$$z_j = \sum_i w_{ij} o_i + b$$

That is the sum over all neurons in the previous layer with their corresponding output o_i and weight w_{ij} towards neuron j plus a bias b.

Now, to update a weight w_{ij} that connects a neuron j in the output layer with a neuron i in the previous layer, I need to calculate the partial derivative of the error function using the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

with z_j as the input to neuron j.

The last term is quite simple. Since there's only one weight between i and j, the derivative is:

$$\frac{\partial z_j}{\partial w_{ij}} = o_i$$

The first term is the derivation of the error function with respect to the output o_j:

$$\frac{\partial E}{\partial o_j} = \frac{-t_j}{o_j}$$

The middle term is the derivation of the softmax function with respect to its input z_j is harder:

Let's say we have three output neurons corresponding to the classes a, b, c then $o_b = softmax(b)$ is:

$$o_b = \frac{e^{z_b}}{\sum e^z} = \frac{e^{z_b}}{e^{z_a} + e^{z_b} + e^{z_c}}$$

and its derivation using the quotient rule:

$$\frac{\partial o_b}{\partial z_b} = \frac{e^{z_b} * \sum e^z - (e^{z_b})^2}{(\sum_j e^z)^2} = \frac{e^{z_b}}{\sum e^z} - \frac{(e^{z_b})^2}{(\sum e^z)^2}$$

$$= softmax(b) - softmax^2(b) = o_b - o_b^2 = o_b(1 - o_b)$$

Back to the middle term for backpropagation this means:

$$\frac{\partial o_j}{\partial z_j} = o_j(1 - o_j)$$

Putting it all together I get

$$\frac{\partial E}{\partial w_{ij}} = \frac{-t_j}{o_j} * o_j(1 - o_j) * o_i = -t_j(1 - o_j) * o_i$$

which means, if the target for this class is $t_j = 0$, then I will not update the weights for this. That does not sound right.

Investigating on this I found people having two variants for the softmax derivation, one where $i = j$ and the other for $i \neq j$, like here or here.

But I can't make any sense out of this. Also I'm not even sure if this is the cause of my error, which is why I'm posting all of my calculations. I hope someone can clarify me where I am missing something or going wrong.

backpropagation derivative softmax cross-entropy

edited Apr 13 '17 at 12:19
Community ♦
1

asked Sep 17 '16 at 23:32
micha
503 1 5 5

The links you have given are calculating the derivative relative to the input, whilst you're calculating the derivative relative to the weights. – Jenkar Sep 18 '16 at 11:25

6 Answers

35

Note: I am not an expert on backprop, but now having read a bit, I think the following caveat is appropriate. When reading papers or books on neural nets, it is not uncommon for derivatives to be written using a mix of the standard summation/index notation, matrix

I noticed a couple of inconsistencies in your derivation. I do not do neural networks really, so the following may be incorrect. However, here is how I would go about the problem.

First, you need to take account of the summation in E, and you cannot assume each term only depends on one weight. So taking the gradient of E with respect to component k of z, we have

$$E = -\sum_j t_j \log o_j \implies \frac{\partial E}{\partial z_k} = -\sum_j t_j \frac{\partial \log o_j}{\partial z_k}$$

Then, expressing o_j as

$$o_j = \tfrac{1}{\Omega} e^{z_j} \, , \, \Omega = \sum_i e^{z_i} \implies \log o_j = z_j - \log \Omega$$

we have

$$\frac{\partial \log o_j}{\partial z_k} = \delta_{jk} - \frac{1}{\Omega} \frac{\partial \Omega}{\partial z_k}$$

where δ_{jk} is the Kronecker delta. Then the gradient of the softmax-denominator is

$$\frac{\partial \Omega}{\partial z_k} = \sum_i e^{z_i} \delta_{ik} = e^{z_k}$$

which gives

$$\frac{\partial \log o_j}{\partial z_k} = \delta_{jk} - o_k$$

or, expanding the log

$$\frac{\partial o_j}{\partial z_k} = o_j(\delta_{jk} - o_k)$$

Note that the derivative is with respect to z_k, an *arbitrary* component of z, which gives the δ_{jk} term ($=1$ only when $k=j$).

So the gradient of E with respect to z is then

$$\frac{\partial E}{\partial z_k} = \sum_j t_j (o_k - \delta_{jk}) = o_k \left(\sum_j t_j \right) - t_k \implies \frac{\partial E}{\partial z_k} = o_k \tau - t_k$$

where $\tau = \sum_j t_j$ is constant (for a given t vector).

This shows a first difference from your result: the t_k no longer multiplies o_k. Note that for the typical case where t is "one-hot" we have $\tau = 1$ (as noted in your first link).

A second inconsistency, if I understand correctly, is that the "o" that is input to z seems unlikely to be the "o" that is output from the softmax. I would think that it makes more sense that this is actually "further back" in network architecture?

$$z_k = \sum_i w_{ik} y_i + b_k \implies \frac{\partial z_k}{\partial w_{pq}} = \sum_i y_i \frac{\partial w_{ik}}{\partial w_{pq}} = \sum_i y_i \delta_{ip} \delta_{kq} = \delta_{kq} y_p$$

Finally, to get the gradient of E with respect to the weight-matrix w, we use the chain rule

$$\frac{\partial E}{\partial w_{pq}} = \sum_k \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial w_{pq}} = \sum_k (o_k \tau - t_k) \delta_{kq} y_p = y_p (o_q \tau - t_q)$$

giving the final expression (assuming a one-hot t, i.e. $\tau = 1$)

$$\frac{\partial E}{\partial w_{ij}} = y_i (o_j - t_j)$$

where y is the input on the lowest level (of your example).

So this shows a second difference from your result: the "o_i" should presumably be from the level below z, which I call y, rather than the level above z (which is o).

Hopefully this helps. Does this result seem more consistent?

> **Update:** In response to a query from the OP in the comments, here is an expansion of the first step. First, note that the vector chain rule requires summations (see here). Second, to be certain of getting *all* gradient components, you should always introduce a *new* subscript letter for the component in the denominator of the partial derivative. So to fully write out the gradient with the full chain rule, we have
>
> $$\frac{\partial E}{\partial w_{pq}} = \sum_i \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial w_{pq}}$$
>
> and
>
> $$\frac{\partial o_i}{\partial w_{pq}} = \sum_k \frac{\partial o_i}{\partial z_k} \frac{\partial z_k}{\partial w_{pq}}$$
>
> so
>
> $$\frac{\partial E}{\partial w_{pq}} = \sum_i \left[\frac{\partial E}{\partial o_i} \left(\sum_k \frac{\partial o_i}{\partial z_k} \frac{\partial z_k}{\partial w_{pq}} \right) \right]$$
>
> In practice the full summations reduce, because you get a lot of δ_{ab} terms. Although it involves a lot of perhaps "extra" summations and subscripts, using the full chain rule will ensure you always get the correct result.

edited Sep 19 '16 at 17:46

answered Sep 18 '16 at 1:34

GeoMatt22

10.7k 2 25 55

I am not certain how the "Backprop/AutoDiff" community does these problems, but I find any time I try to take shortcuts, I am liable to make errors. So I end up doing as here, writing everything out in terms of

Although I'm still trying to fully understand each of your steps, I got some valuable insights that helped me with the overall picture. I guess I need to read more into the topic of derivations and sums. But taking your advise to take account of the summation in E, I came up with this: – micha Sep 19 '16 at 6:54

for two outputs $o_{j_1} = \frac{e^{z_{j_1}}}{\Omega}$ and $o_{j_1} = \frac{e^{z_{j_1}}}{\Omega}$ with

$$\Omega = e^{z_{j_1}} + e^{z_{j_2}}$$

the cross entropy error is

$$E = -(t_1 log o_{j_1} + t_2 log o_{j_2}) = -(t_1(z_{j_1} - log(\Omega)) + t_2(z_{j_2} - log(\Omega)))$$

Then the derivative is

$$\frac{\partial E}{\partial (z_{j_1}} = -(t_1 - t_1 \frac{e^{z_{j_1}}}{\Omega} - t_2 \frac{e^{z_{j_2}}}{\Omega}) = -t_1 + o_{j_1}(t_1 + t_2)$$

which conforms with your result... taking in account that you didn't have the minus sign before the error sum – micha Sep 19 '16 at 7:10

But a further question I have is: Instead of

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

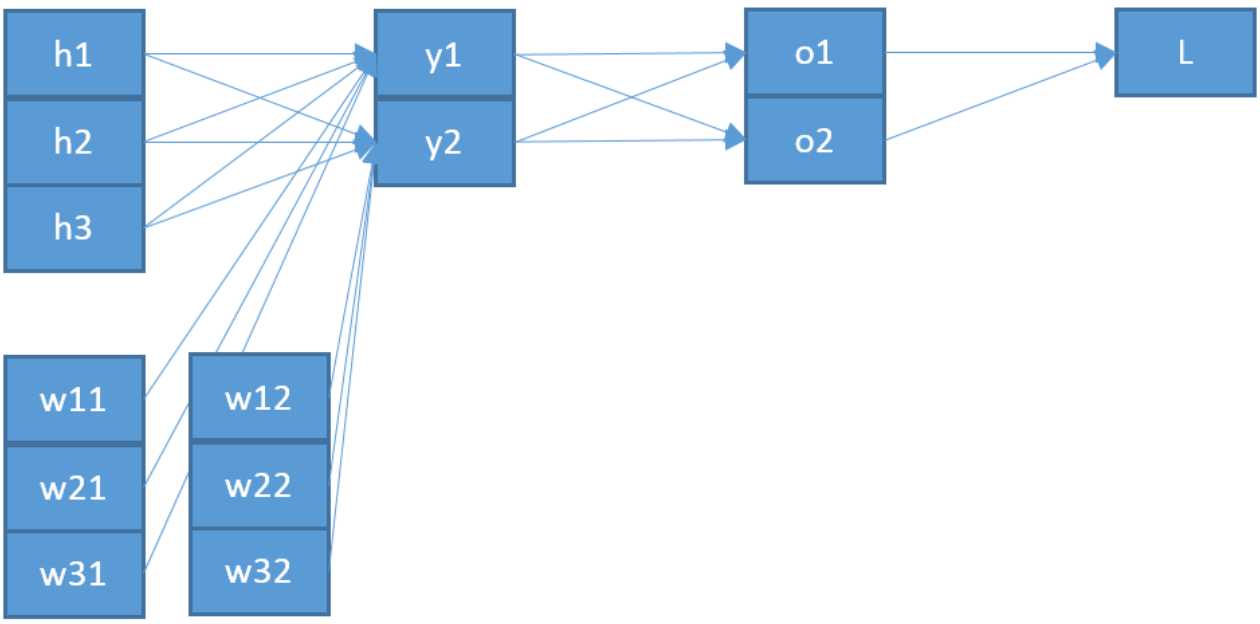
which is generally what your introduced to with backpropagation, you calculated:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

as like to cancel out the ∂o_j . Why is this way leading to the right result? – micha Sep 19 '16 at 7:20

12

While @GeoMatt22's answer is correct, I personally found it very useful to reduce the problem to a toy example and draw a picture:



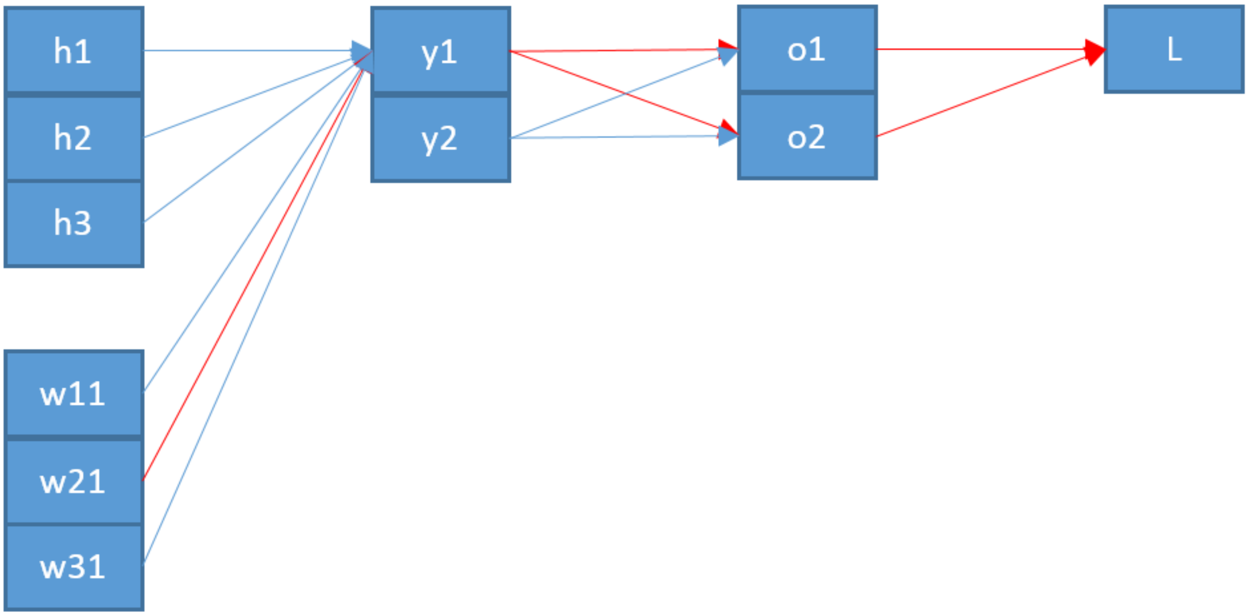
$$o_1 = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}$$

$$o_2 = \frac{\exp(y_2)}{\exp(y_1) + \exp(y_2)}$$

$$y_1 = w_{11}h_1 + w_{21}h_2 + w_{31}h_3$$

$$y_2 = w_{12}h_1 + w_{22}h_2 + w_{32}h_3$$

Say I want to calculate the derivative of the loss with respect to w_{21}. I can just use my picture to trace back the path from the loss to the weight I'm interested in (removed the second column of w's for clarity):



Then, I can just calculate the desired derivatives. Note that there are two paths through y_1 that lead to w_{21}, so I need to sum the derivatives that go through each of them.

$$\frac{\partial L}{\partial o_1} = -\frac{t_1}{o_1}$$

$$\frac{\partial L}{\partial o_2} = -\frac{t_2}{o_2}$$

$$\frac{\partial o_1}{\partial y_1} = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)} - \left(\frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}\right)^2 = o_1(1 - o_1)$$

$$\frac{\partial o_2}{\partial y_1} = \frac{-\exp(y_2)\exp(y_1)}{(\exp(y_1) + \exp(y_2))^2} = -o_2 o_1$$

$$\frac{\partial y_1}{\partial w_{21}} = h_2$$

Finally, putting the chain rule together:

$$\begin{aligned}\frac{\partial L}{\partial w_{21}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} \\ &= \frac{-t_1}{o_1}[o_1(1-o_1)]h_2 + \frac{-t_2}{o_2}(-o_2 o_1)h_2 \\ &= h_2(t_2 o_1 - t_1 + t_1 o_1) \\ &= h_2(o_1(t_1+t_2) - t_1) \\ &= h_2(o_1 - t_1)\end{aligned}$$

Note that in the last step, $t_1 + t_2 = 1$ because the vector $\mathbf{t}$ is a one-hot vector.

edited Mar 31 '18 at 2:43 answered Oct 7 '17 at 5:26

Vivek Subramanian
1,613 1 14 23

This is what finally cleared this up for me! Excellent and Elegant explanation!!!! – SantoshGupta7 Feb 16 '18 at 14:08

2 I'm glad you both enjoyed and benefited from reading my post! It was also helpful for me to write it out and explain it. – Vivek Subramanian Feb 16 '18 at 14:19

@VivekSubramanian should it be

$$= \frac{-t_1}{o_1}[o_1(1-o_1)]h_2 + \frac{-t_2}{o_2}(-o_2 o_1)h_2$$

instead ? – koryakinp Mar 31 '18 at 2:40

You're right - it was a typo! I will make the change. – Vivek Subramanian Mar 31 '18 at 2:42

The thing i do not understand here is that you also assign logits (unscaled scores) to some neurons. (o is softmaxed logits (predictions) and y is logits in your case). However, this is not the case normally, is not it? Look at this picture (o_out1 is prediction and o_in1 is logits) so how is it possible in this case how can you find the partial derivative of o2 with respect to y1? – ARAT Mar 11 at 16:43

6

In place of the $\{o_i\}$, I want a letter whose uppercase is visually distinct from its lowercase. So let me substitute $\{y_i\}$. Also, let's use the variable $\{p_i\}$ to designate the $\{o_i\}$ from the previous layer.

Let Y be the diagonal matrix whose diagonal equals the vector y, i.e.

$$Y = \mathrm{Diag}(y)$$

Using this new matrix variable and the Frobenius Inner Product we can calculate the gradient of E wrt W.

$$\begin{aligned} z &= Wp + b & dz &= dW p \\ y &= \text{softmax}(z) & dy &= (Y - yy^T)\, dz \\ E &= -t : \log(y) & dE &= -t : Y^{-1} dy \end{aligned}$$

$$\begin{aligned} dE &= -t : Y^{-1}(Y - yy^T)\, dz \\ &= -t : (I - 1y^T)\, dz \\ &= -t : (I - 1y^T)\, dW\, p \\ &= (y1^T - I)tp^T : dW \\ &= ((1^T t)yp^T - tp^T) : dW \end{aligned}$$

$$\frac{\partial E}{\partial W} = (1^T t)yp^T - tp^T$$

answered Sep 20 '16 at 3:09
frank
61 1

6

Here is one of the cleanest and well written notes that I came across the web which explains about **"calculation of derivatives in backpropagation algorithm with cross entropy loss function"**.

answered Jul 10 '17 at 6:37
yottabytt
161 1 2

In the given pdf how did equation 22 become equation 23? As in how did the Summation(k!=i) get a negative sign. Shouldn't it get a positive sign? Like `Summation(Fn)(For All K) = Fn(k=i) + Summation(Fn)(k!=i)` should be happening according to my understanding. – faizan Jan 13 '18 at 6:59

1

Here's a link explaining the softmax and its derivative.

It explains the reason for using i=j and i!=j.

answered Jun 18 '17 at 6:54
S. Muhammad H. Mustafa
11 2

It is recommended to provide a minimal, stand-alone answer, in case that link gets broken in the future. Otherwise, this might no longer help other users in the future. – luchonacho Jun 18 '17 at 8:55

0

assumed that o_j takes that node of target 1 in your output; o_j of other nodes have different forms of probability function, thus lead to different forms of derivative, so you should now understand why other people have treated $i = j$ and $i \neq j$ differently.

answered Sep 23 at 13:38

kuixiong

1