2 Comments / Mathematics of Machine Learning / By mehran@mldawn.com

What will you learn?

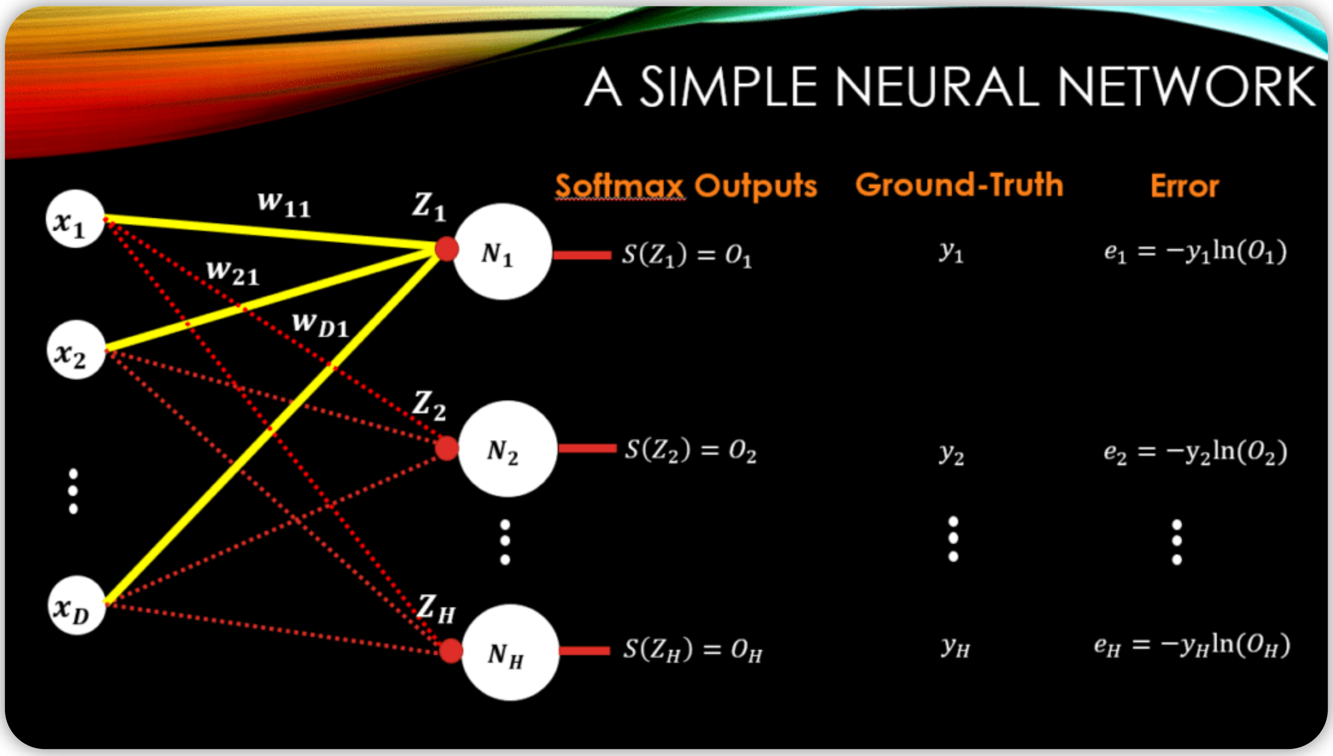
This post is also available to you in this video, should you be interested 😉

In our previous post, we talked about the derivative of the **softmax function with respect to its input**. We indeed beautifully dissect ed the math and got comfortable with it! In this post, we will go one step further! Let's say you have a neural network with softmax output layer, and you are using the cross-entropy error function. Today, we will derive the gradient of the cross-entropy error with respect to the input of the softmax function.

This is one of the most confusing mathematical derivations that machine learning enthusiasts tend to have problems with. This post is important as it shows you, if you have a softmax output layer, and use the cross-entropy function, how you can apply back-propagation from the error all the way to the input of the softmax layer, which is the tricky part of the entire back-propagation in this type of neural networks.

A Typical Neural Network

Let's consider a simple neural network with D-dimensional input data, and H output neurons with softmax output function. So, for H output neurons, we will have H softmax outputs. We will have one-hot encoding ground-truth vectors and finally we will have the cross-entropy error function to compute the distance between the output vector and the ground-truth vector. This is a typical neural network for a multi-class classification task. So, for a given training example as the input, the network will generate an output vector with the size of H(which is the number of classes in our dataset)!



So, just as a gentle reminder, the softmax function can be defined as:

$$S(Z_k)=\frac{e^{Z_k}}{\sum_{i=1}^{H}e^{Z_i}}$$

And the interesting property of this function is that the sum of all the outputs of softmax, is always equal to 1:

$$\sum_{i=1}^{H}S(Z_i)=1$$

Now, about the ground-truth vector, y, we mentioned that it is a one-hot encoding vector. This means that for every ground-truth vector **ONLY** one element can be equal to 1 and all the other elements are equal to 0!

Finally, regarding the cross-entropy error function, the mathematical representation of this function is as follows:

$$E(y,S(Z))=-\sum_{i=1}^{H}y_i\times ln(S(Z_i))$$

In this definition of error, ln, is the natural logarithm. As for an example, let's say we have 3 output neurons. For a given training example, the output vector of this neural network will have 3 elements in it. Let's say the output vector is as follows:

$$S(Z)=[S(Z_1),S(Z_2),S(Z_3))]=[0.2,0.6,0.2]$$

You notice that these sum up to 1 as the property of softmax function. And, let's say the ground-truth vector for the same input training example is as follows:

$$y=[y_1,y_2,y_3]=[1,0,0]$$

With s simple comparison between the network output vector and the ground-truth vector you can see that, the network thinks that the given training example belongs to class 2 (as 0.6 is the largest value and corresponds to class 2), however, the ground-truth says that the training example actually belongs to class 1 (as only the value corresponding to the first class is 1 and all the others are 0).

Now, let's see how we can compute the cross-entropy error function:

$$E(y,S(Z)) = -(y_1 \times ln(S(Z_1)) + y_2 \times ln(S(Z_2)) + y_3 \times ln(S(Z_3)))$$

which is equal to:

$$E(y,S(Z)) = -(1 \times ln(0.2) + 0 \times ln(0.6) + 0 \times ln(0.2)) = -ln(0.2)$$

which is:

$$E(y,S(Z)) = 1.6094$$

Now that we are comfortable with the whole setting, let's see how we can derive the gradient of this error function with respect to the inputs of the softmax output function and apply back-propagation from scratch!

Deriving Back-propagation through Cross-Entropy and Softmax

In order to fully understand the back-propagation in here, we need to understand a few mathematical rules regarding partial derivatives.

Rule 1) Derivative of a SUM is equal to the SUM of derivatives

Rule 2) The rule of Independence

$$\frac{\partial[f(x)+g(x)]}{\partial x} = \frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$$

$$\frac{\partial[f(x) \times g(y)]}{\partial x} = g(y) \times \frac{\partial f(x)}{\partial x}$$

Rule 3) The Chain Rule

if u is a function of x (i.e., $u(x)$) and f is a function of u (i.e., $f(u)$) then:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \times \frac{\partial u}{\partial x}$$

Rule 4) Derivative of the log function

$$\frac{\partial log_b(x)}{\partial x} = \frac{1}{ln(b) \times x}$$

You can see that we could easily extend **Rule #4** to the natural logarithm where the base of the logarithm is the Euler's number (i.e., 2.7182 …):

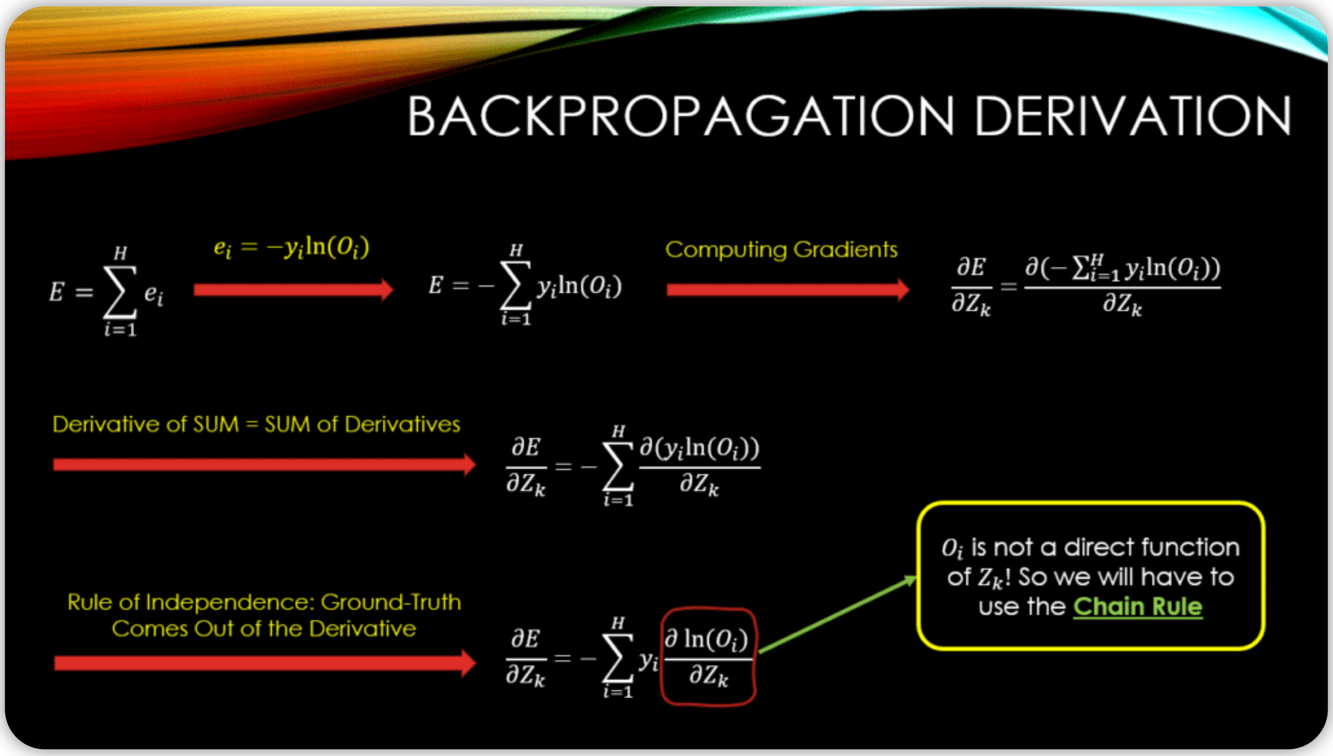
$$\frac{\partial ln(x)}{\partial x} = \frac{1}{ln(e) \times x} = \frac{1}{x}$$

This happens because $ln(e) = 1$.

In order to derive the back-propagation math, we will first have to compute the total error across all the output neurons of our neural network and only then can we start with our derivatives and back-propagation. So, we will sum up all the individual errors, e_i, for $i \in \{1,2,...,H\}$, and this will be total error across all output units!

> ***NOTE**: In this example, this total error corresponds to the output of the network for **ONLY** one input training example! Meaning that, if we wanted to compute the error across the whole training examples, we should add another summation to our error computation:* $\sum_{d \in D} \sum_{i=1}^{H} e_{di}$ *where D is the entire training set. In here,* e_{di} *corresponds to the error value in the* i^{th} *output neuron when the* d^{th} *training example is fed to the neural network.*

Now, in order to apply back-propagation, we will have to compute the gradients of the total error w.r.t the inputs of the softmax layer. Please note that this is only the beginning of the back-propagation! If you had 20 layers in your neural network, you would have to compute the gradient of your error w.r.t all the learnable parameters across all 20 layers! However, I believe this initial part is the step where most people are not quite comfortable with. Now, let's see how we can compute the gradient of the total error w.r.t the inputs of the softmax layer:



BACKPROPAGATION DERIVATION

$$E = \sum_{i=1}^{H} e_i \xrightarrow{e_i = -y_i \ln(O_i)} E = -\sum_{i=1}^{H} y_i \ln(O_i) \xrightarrow{\text{Computing Gradients}} \frac{\partial E}{\partial Z_k} = \frac{\partial(-\sum_{i=1}^{H} y_i \ln(O_i))}{\partial Z_k}$$

Derivative of SUM = SUM of Derivatives

$$\longrightarrow \frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H} \frac{\partial(y_i \ln(O_i))}{\partial Z_k}$$

Rule of Independence: Ground-Truth Comes Out of the Derivative

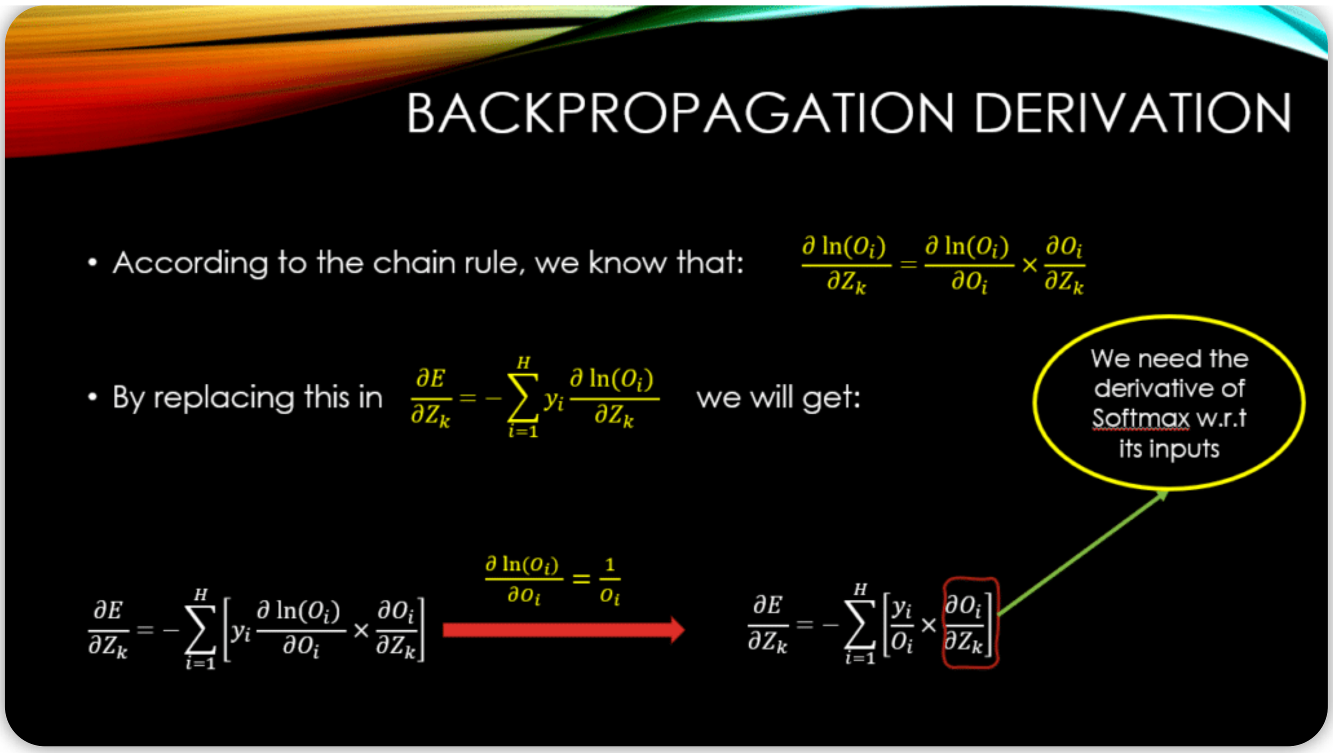
$$\longrightarrow \frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H} y_i \boxed{\frac{\partial \ln(O_i)}{\partial Z_k}}$$

O_i is not a direct function of Z_k! So we will have to use the Chain Rule

You can see that we have subtly used **Rule #1**, in order to take the partial derivative operation into our SUM! And we have also grabbed our ground-truth variable y_i out of the partial derivative operation, using **Rule #2** of independence! Because our ground-truth variable is independent of all the inputs to our softmax layer, Z_k! As a result, we can treat it as an independent variable from

Z_k, and bring it out of the partial derivative operation! Finally you notice that we will have to use **Rule #3**, that is the chain rule!

> ***Why the chain rule?*** *Well because in* $\frac{\partial ln(O_i)}{\partial Z_k}$, *we notice that* $ln(O_i)$ *is* ***not*** *a direct function of* Z_k*! However, it is a direct function of* O_i *! In turn,* O_i *is indeed a direct function of* Z_k*! So, in order to compute* $\frac{\partial ln(O_i)}{\partial Z_k}$, *we will have to go through* O_i *first and then get to* Z_k*! By chaining these steps together we will have our chain!!!*



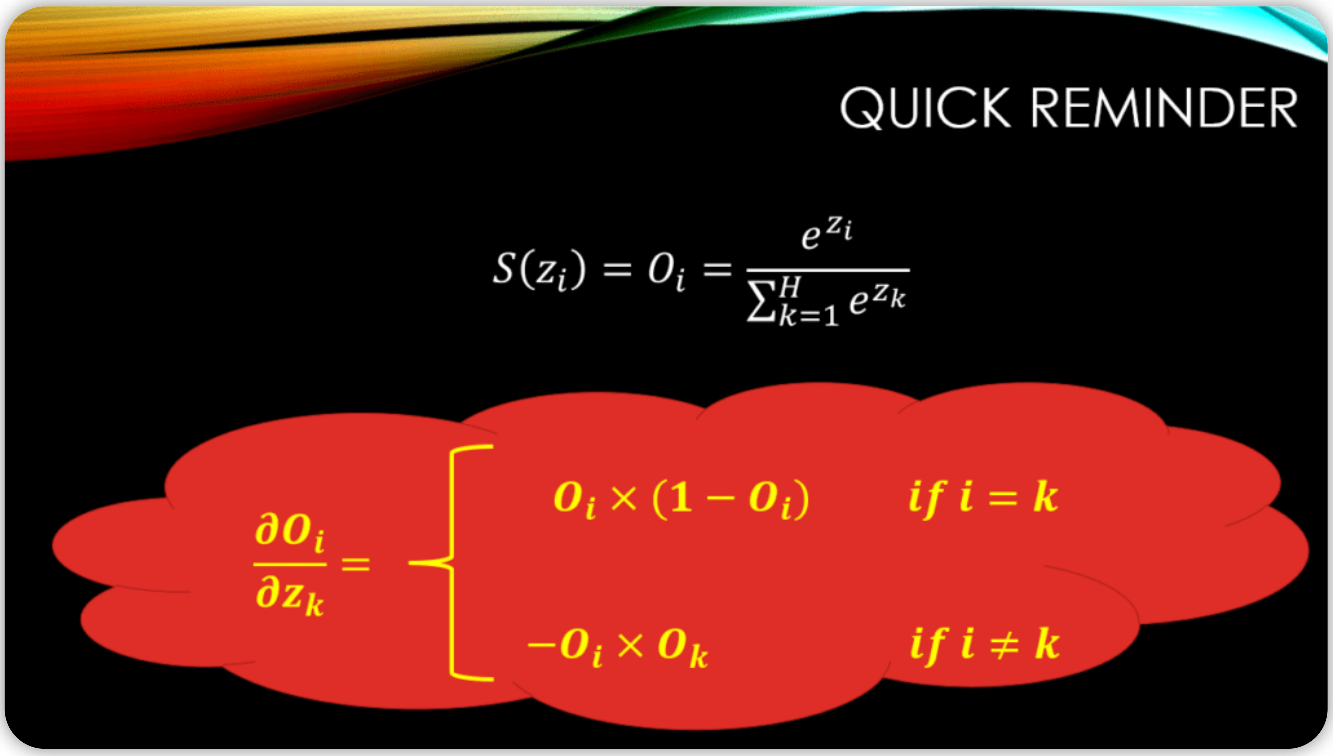
BACKPROPAGATION DERIVATION

- According to the chain rule, we know that: $\frac{\partial \ln(O_i)}{\partial Z_k} = \frac{\partial \ln(O_i)}{\partial O_i} \times \frac{\partial O_i}{\partial Z_k}$
- By replacing this in $\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H} y_i \frac{\partial \ln(O_i)}{\partial Z_k}$ we will get:

We need the derivative of Softmax w.r.t its inputs

$$\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}\left[y_i \frac{\partial \ln(O_i)}{\partial O_i} \times \frac{\partial O_i}{\partial Z_k}\right] \xrightarrow{\frac{\partial \ln(O_i)}{\partial O_i} = \frac{1}{O_i}} \frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}\left[\frac{y_i}{O_i} \times \boxed{\frac{\partial O_i}{\partial Z_k}}\right]$$

Now after applying the chain rule, you notice that we have also used **Rule #4** to find the derivative of our natural logarithm, that is, $\frac{\partial ln(O_i)}{\partial O_i}$. We have got to the place where we will need the derivative of the softmax function w.r.t its input (i.e., $\frac{\partial O_i}{\partial Z_k}$). We have already covered the derivative of softmax w.r.t its input in our previous post, but as a reminder, see the slide blow:

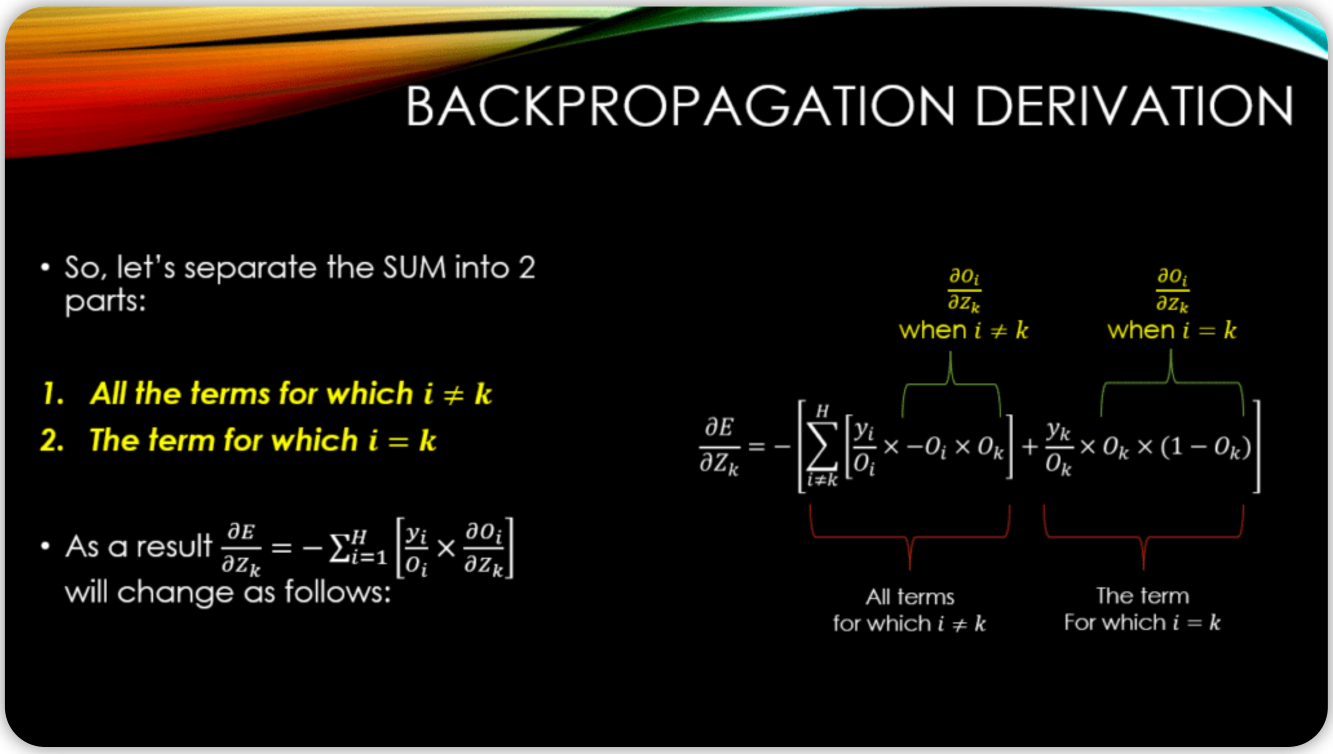


QUICK REMINDER

$$S(z_i) = O_i = \frac{e^{z_i}}{\sum_{k=1}^{H} e^{z_k}}$$

$$\frac{\partial O_i}{\partial z_k} = \begin{cases} O_i \times (1 - O_i) & if\ i = k \\ -O_i \times O_k & if\ i \neq k \end{cases}$$

So, we can see that when it comes to $\frac{\partial O_i}{\partial Z_k}$, it all boils down to whether $i = k$ or $i \neq k$! As a result, in order to compute $\frac{\partial O_i}{\partial Z_k}$, for a given Z_k, we will consider all O_i's for $i \in \{1,2,...,H\}$! There will be **ONLY** one case where $i=k$, and for all other values of i, for sure we will have $i \neq k$! Thus let's solve $\frac{\partial O_i}{\partial Z_k}$ for both cases:



BACKPROPAGATION DERIVATION

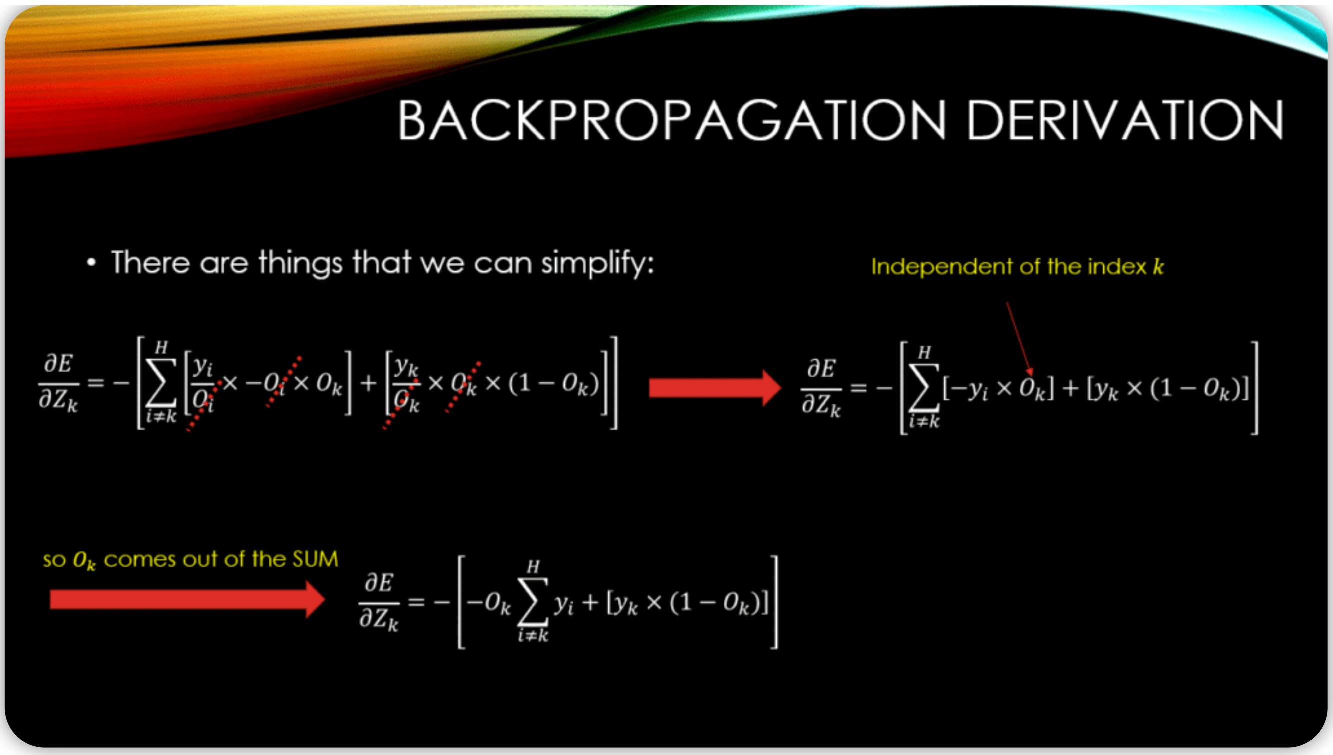
- So, let's separate the SUM into 2 parts:

1. ***All the terms for which $i \neq k$***
2. ***The term for which $i = k$***

- As a result $\frac{\partial E}{\partial Z_k} = -\sum_{i=1}^{H}\left[\frac{y_i}{O_i} \times \frac{\partial O_i}{\partial Z_k}\right]$ will change as follows:

$$\frac{\partial E}{\partial Z_k} = -\left[\underbrace{\sum_{i \neq k}^{H}\left[\frac{y_i}{O_i} \times \overbrace{-O_i \times O_k}^{\frac{\partial O_i}{\partial Z_k}\text{ when } i \neq k}\right]}_{\text{All terms for which } i \neq k} + \underbrace{\frac{y_k}{O_k} \times \overbrace{O_k \times (1 - O_k)}^{\frac{\partial O_i}{\partial Z_k}\text{ when } i = k}}_{\text{The term For which } i = k}\right]$$

Now, the hard part is over! You can see how we have taken out the term for which $i=k$! Then we have used the derivative of softmax (extensively discussed in our previous post) in order to finally be done with the partial derivative operations. Now, let's simplify:



BACKPROPAGATION DERIVATION

- There are things that we can simplify:

$$\frac{\partial E}{\partial Z_k} = -\left[\sum_{i\neq k}^{H}\left[\frac{y_i}{\cancel{O_i}} \times -\cancel{O_i} \times O_k\right] + \left[\frac{y_k}{\cancel{O_k}} \times \cancel{O_k} \times (1 - O_k)\right]\right] \longrightarrow \frac{\partial E}{\partial Z_k} = -\left[\sum_{i\neq k}^{H}[-y_i \times O_k] + [y_k \times (1 - O_k)]\right]$$

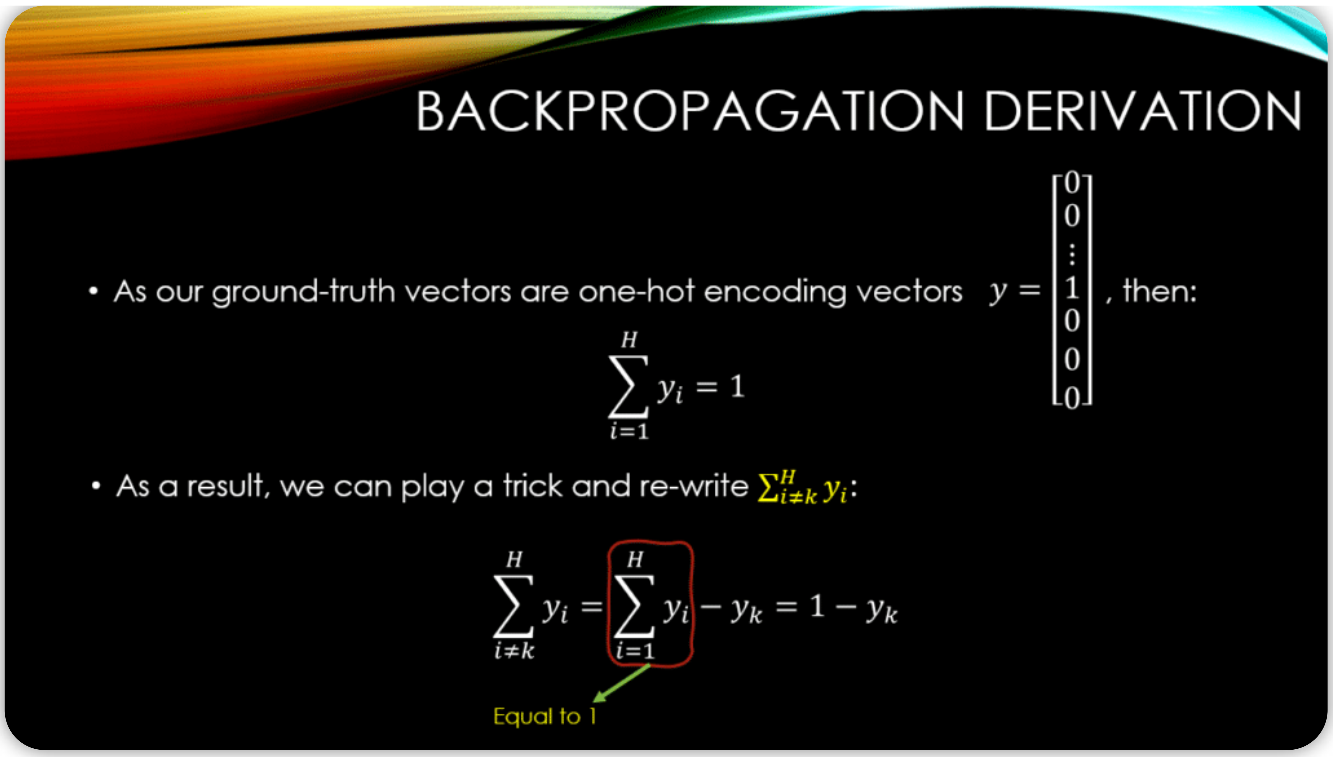
Independent of the index k

so O_k comes out of the SUM

$$\longrightarrow \frac{\partial E}{\partial Z_k} = -\left[-O_k \sum_{i\neq k}^{H} y_i + [y_k \times (1 - O_k)]\right]$$

You notice that O_k has come out of the SUM! The reason is simply because, the index of O_k, is independent of i, that is the index over which we are summing! So, O_k is treated as a constant coefficient of y_i!

In order to get rid of the summing operation, the next trick we can play is taking advantage of the fact that, the ground-truth vector y, is a **one-hot encoding vector**! As a result, if we sum all of its elements across all H outputs, the result will always be equal to 1 (i.e., $\sum_{i=1}^{H} y_i = 1$). So, in order to compute $\sum_{i\neq k}^{H} y_i$, we can subtract the k^{th} element of y (i.e., y_k), from $\sum_{i=1}^{H} y_i$. This is how we will get rid of the SUM from our math! See below:



BACKPROPAGATION DERIVATION

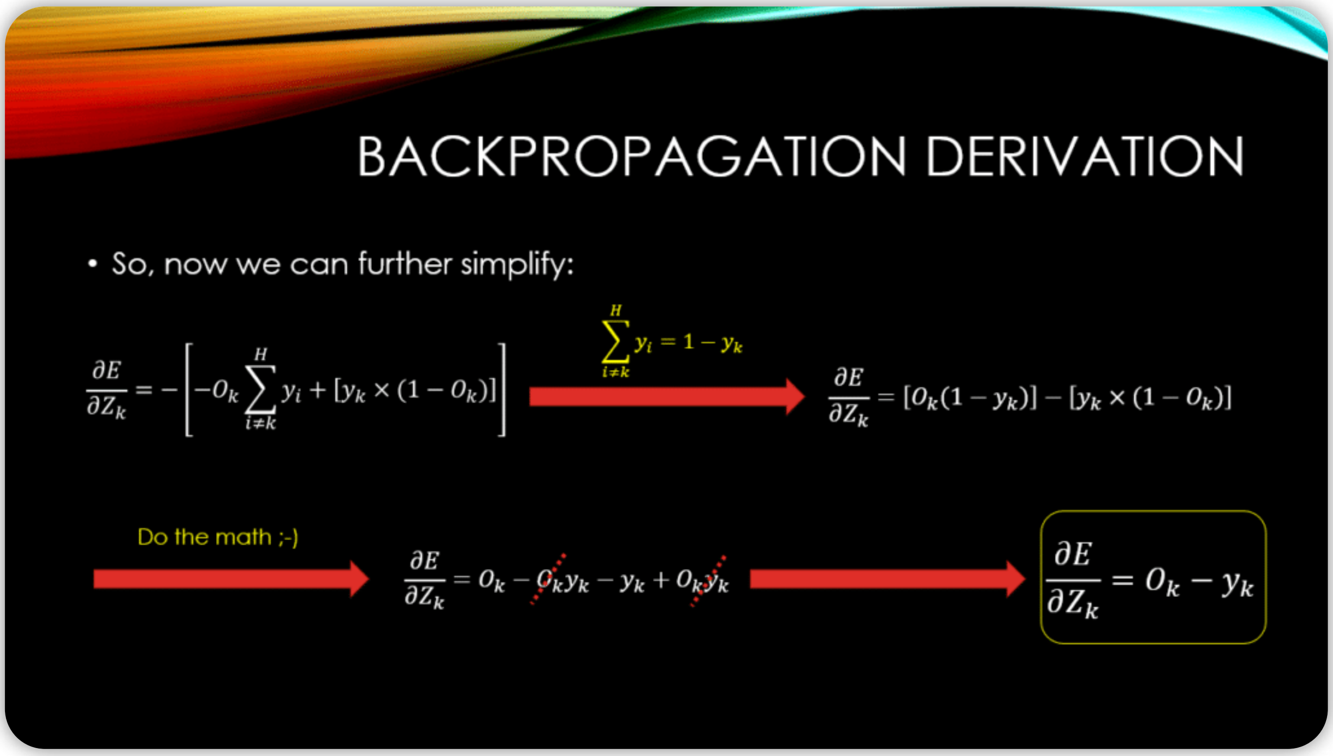
- As our ground-truth vectors are one-hot encoding vectors $y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, then:

$$\sum_{i=1}^{H} y_i = 1$$

- As a result, we can play a trick and re-write $\sum_{i \neq k}^{H} y_i$:

$$\sum_{i \neq k}^{H} y_i = \underbrace{\boxed{\sum_{i=1}^{H} y_i}}_{\text{Equal to 1}} - y_k = 1 - y_k$$

So, now that we have computed $\sum_{i \neq k}^{H} y_i$ with our smart approach, let's replace it in our derivations, and simplify more:



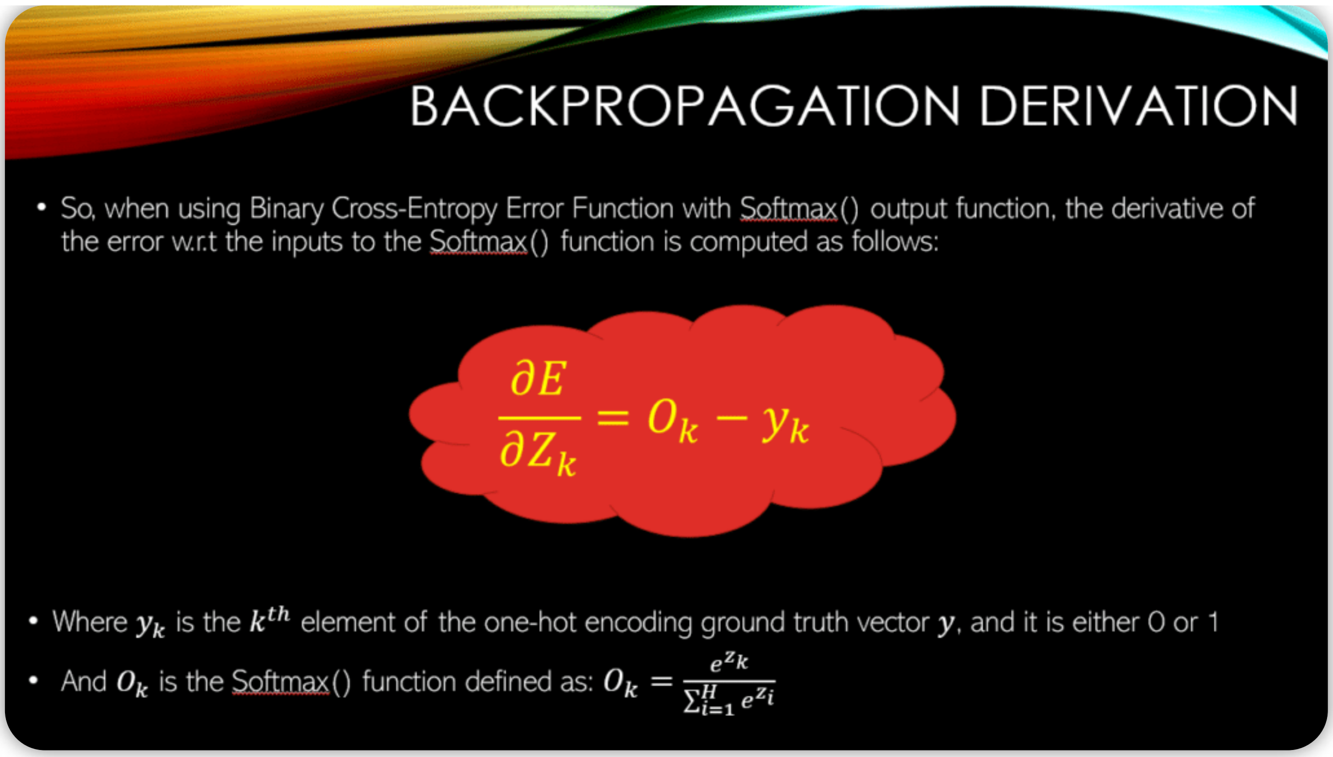
BACKPROPAGATION DERIVATION

- So, now we can further simplify:

$$\frac{\partial E}{\partial Z_k} = -\left[-O_k \sum_{i \neq k}^{H} y_i + [y_k \times (1 - O_k)] \right] \xrightarrow{\sum_{i \neq k}^{H} y_i = 1 - y_k} \frac{\partial E}{\partial Z_k} = [O_k(1 - y_k)] - [y_k \times (1 - O_k)]$$

$$\xrightarrow{\text{Do the math ;-)}} \frac{\partial E}{\partial Z_k} = O_k - \cancel{O_k} y_k - y_k + O_k \cancel{y_k} \longrightarrow \boxed{\frac{\partial E}{\partial Z_k} = O_k - y_k}$$

Perfect! We are finally done! Let's see the final answer in a nice and concise way:



BACKPROPAGATION DERIVATION

- So, when using Binary Cross-Entropy Error Function with Softmax() output function, the derivative of the error w.r.t the inputs to the Softmax() function is computed as follows:

$$\frac{\partial E}{\partial Z_k} = O_k - y_k$$

- Where y_k is the k^{th} element of the one-hot encoding ground truth vector y, and it is either 0 or 1
- And O_k is the Softmax() function defined as: $O_k = \frac{e^{z_k}}{\sum_{i=1}^{H} e^{z_i}}$

Conclusions

So there you have it! Now you can see the very simple outcome of this whole monstrous series of derivations! The fact that it could be simplified to this level, is the benefit of using the cross-entropy error function with softmax output function in neural networks! However, this simplicity is not the main reason as to why we use softmax coupled with cross-entropy error function, mind you! This is just the nice outcome that we can enjoy!

Until next time, on behalf of MLDawn

Take care 😉

← Previous Post Next Post →

2 thoughts on “Back-propagation with Cross-Entropy and Softmax”

MOIN
NOVEMBER 24, 2021 AT 2:06 PM

Great tutorial and explanation! Only thing i would fix are the images (equations) showing the rules of partial derivatives. They are overlapping the text

Reply

MEHRAN@MLDAWN.COM
NOVEMBER 24, 2021 AT 4:43 PM

Thanks for the feedback. Duely noted!

Reply

Leave a Comment

Your email address will not be published.

Type here..

Name*

Email*

Website

Post Comment »

Please follow & like us :)

Search ...

Recent Posts

A Gentle 101 Talk on Artificial Neural Networks

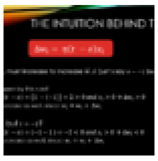
The Backpropagation Algorithm-PART(1): MLP and Sigmoid

Dissecting Relu: A desceptively simple activation function

An interview with Prof. Mikhail Belkin

Reconciling modern machine learning practice and the bias-variance trade-off

Most Popular Posts



Theory and Concepts

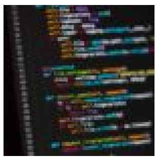
The Perceptron Training Rule

Mathematics of Machine Learning

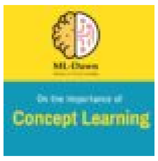
Back-propagation with Cross-Entropy and Softmax

Mathematics of Machine Learning

The Derivative of Softmax(z) Function w.r.t z



Coding
Train a Perceptron to Learn the AND Gate from Scratch in Python



Theory and Concepts
Concept Learning and General to Specific Ordering-Part(3)

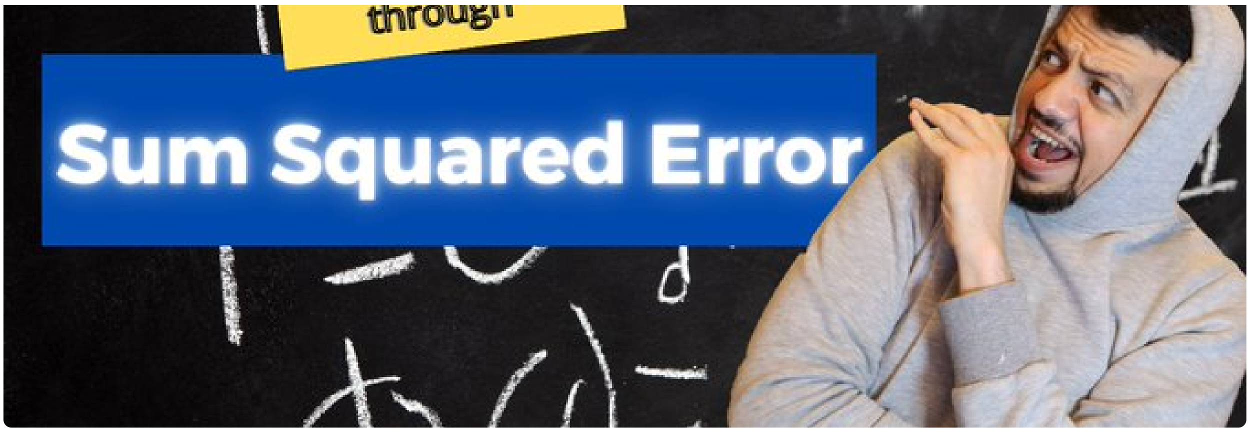
Categories

- Algorithms
- Coding
- Interviews
- Mathematics of Machine Learning
- Paper Analysis
- Public Talks
- Talks
- Theory and Concepts

Tweet Posts

Tweets by @MLDawn2018

ML-Dawn
@MLDawn2018
We #NeuralNetworks for #regression with Sum Square Error function and Linear output units. In this video, we do back-propagation through SSE and these linear units from scratch:youtu.be/d9AvALaC-5s#machinelearning #deeplearning #Maths @insight_centre @scienceirel @UCDCompSci



14h

ML-Dawn Retweeted

UCD Computer Science
@UCDCompSci

Outstanding opportunity: Assistant Professor positions in UCD School of Computer Science (apply on Work at UCD website ucd.ie/workatucd/jobs/ by 17 Jan, ref: 013992). More info:universityvacancies.com/university-col…

Dec 15, 2021

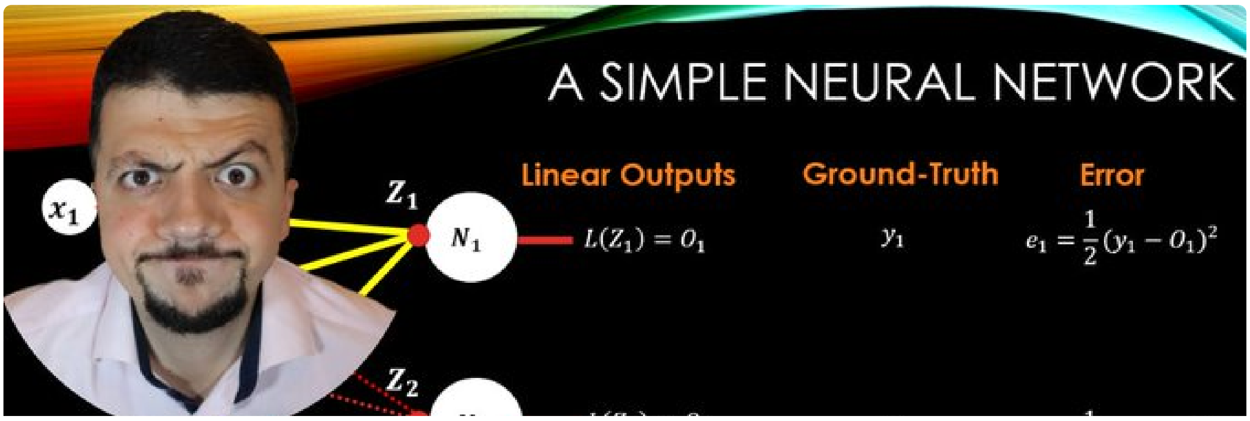
ML-Dawn
@MLDawn2018

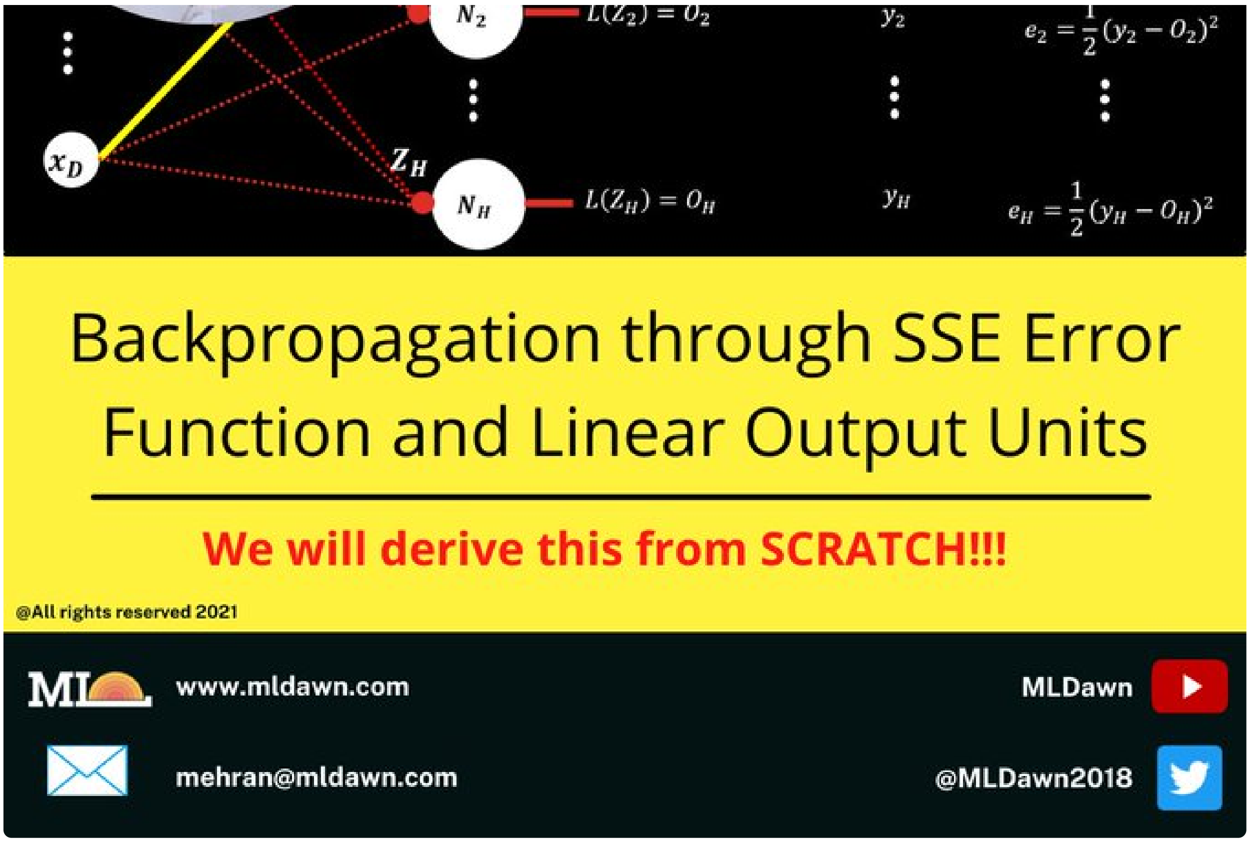
Here we go! Cannot wait to attend this!
https://twitter.com/AicsConference/status/1468718591881236483

Dec 9, 2021

ML-Dawn
@MLDawn2018

#machinelearningalgorithms #deeplearning
In #machinelearning, when we use #neuralnetworks for #regression , we need the Sum Square Error (SSE) function and Linear outputs. I will soon show you how backpropagation passes your gradients through SSE and the Linear outputs.




N_2
$L(Z_2) = O_2$
y_2
$e_2 = \frac{1}{2}(y_2 - O_2)^2$
x_D
Z_H
N_H
$L(Z_H) = O_H$
y_H
$e_H = \frac{1}{2}(y_H - O_H)^2$
Backpropagation through SSE Error Function and Linear Output Units
We will derive this from SCRATCH!!!
@All rights reserved 2021
www.mldawn.com
MLDawn
mehran@mldawn.com
@MLDawn2018

Dec 7, 2021

ML-Dawn
@MLDawn2018

Let the countdown begin!
https://twitter.com/AicsConference/status/1466153290572390409

About Blog Contact Home Interviews Public Talks