Finding gradients of neural networks

Stanley Neoh

May 2023

1 Introduction

This article aims to cover the basic mathematics of neural networks. neural networks are basically fancy functions with many parameters that aim to fit some set of data. The training of neural networks consists of *forward propagation* where output is produced from a given input and *backward propagation* where the output is compared to its corresponding labels and the parameters are tweaked accordingly to better fit.

The most challenging aspect of neural networks for me personally was to stray away from thinking of tensors and vectors as rows and columns. This made it very difficult to calculate the gradients for high dimensional tensors. Hence, this short article aims to establish a set of notations that you may adopt to better understand the mathematics of neural networks.

2 'Einstan' notation

As we often will be differentiating high dimensional tensors with respect to other high dimensional tensors, we will be using a modified variant of Einstein Notation. Just because I think it's funny, let's call this notation 'Einstan'[1] notation.

Any rank-n tensor A can be represented in the following:

$$A = A_{i_1,i_2,\ldots,i_n}$$

To prevent ourselves from being bogged down by the concept of rows and columns, we will avoid talking about a tensor in terms of its shape. We will primarily be recognizing the axis from its ordering. In the above example, the i_1 axis is the first axis, the i_2 axis is the second axis, and so on.

[1] Einstan is a merger of the words Einstein and Stanley which is my name.

3 Inner product with 'Einstan' notation

The inner product of 2 vectors u and v can be expressed as:

$$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

In 'Einstan' notation, the same inner product is expressed as

$$u_i v_i$$

Notice that the summation sign is omitted. So how do we tell which index to sum across? That's simple, so long as there is a common index shared between 2 tensors **in a product**, we will sum across it implicitly unless otherwise stated. This is very similar to Einstein's sum notation except that we do not require one of the indexes to be a superscript and the other to be a subscript. Everything is subscript in our notation.

Let us consider the inner product of a rank-2 tensor $A = A_{i,j}$ with a rank-1 column vector $b = b_k$. We could perform the inner-product along the first axis of A by arranging the A and b in the following way:

$$b^T A = A_{k,j} b_k$$

We could also perform the inner-product along the second axis of A by arranging A and b in the following way:

$$Ab = A_{i,k} b_k$$

This shows us that the inner product can happen along any single axis between 2 tensors. This can be generalized and expressed in our notation in the following way:

$$A \cdot B = A_{i_1,i_2,\ldots,i_a=k,\ldots,i_n} B_{j_1,j_2,\ldots,j_b=k,\ldots,j_m}$$

This would mean that the inner product of A with that of B happens along the a-th axis of A and the b-th axis of B. Perhaps a higher dimensional being could write it out in the standard matrix notation, but with our limited dimensional thinking, this is probably one of the better options we have.

Why stop there? Why does it have to be along 1 axis only? Can't we perform inner product across multiple axes? This may be useful in the future, so we will define our generalized inner product to do this as well (since our notation can easily handle this case anyways).

Now armed with this compact notation, let's see if we can concisely tackle multi-variable differentiation.

4 Differentiating tensors w.r.t. tensors

We are familiar with differentiating a scalar-valued function with respect to a scalar. That would look like the following:

$$\frac{d}{dx}(f(x)) = \frac{df}{dx}$$

That was simple enough, how about differentiating a vector-valued function with respect to another scalar? We could perform partial differentiation with respect to each variable and write the final result out as a vector like so:

$$\frac{\partial f}{\partial (x,y)} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}$$

Some mathematicians like to put $\frac{\partial f}{\partial (x,y)}$ as a column vector, while others like to put it as a row vector. This isn't really important as it is up to personal preference. We will see that with 'Einstan' notation, this issue of deciding between rows or columns is irrelevant.

How about a vector-valued function with a scalar? This is also straightforward. Just simply differentiate each component and put it into a vector like so:

$$\frac{dF}{dx} = \begin{bmatrix} \frac{dF_x}{dx} \\ \frac{dF_y}{dx} \end{bmatrix}$$

Differentiating a vector-valued function with a vector is just combining the 2 scenarios above like so:

$$\frac{\partial F}{\partial (x,y)} = \begin{bmatrix} \frac{dF_x}{dx} & \frac{dF_x}{dy} \\ \frac{dF_y}{dx} & \frac{dF_y}{dy} \end{bmatrix}$$

But things start to get difficult to write down when we start going to larger ranks. So let's try not to write it down this way then. The purpose of having rows and columns is to establish the axes the inner product is performed along which is taken care of by our notation.

When differentiating a rank-n tensor A and rank-m tensor B, we will just write it down as so:

$$\frac{\partial A}{\partial B} = \frac{\partial A_{i_1,i_2,\ldots,i_n}}{\partial B_{j_1,j_2,\ldots,j_m}}$$

where $\frac{\partial A}{\partial B}$ is a tensor of rank $n+m$.

5 Chain rule with 'Einstan' notation

Chain rule is a crucial part of backpropagation. So let's see what the chain rule looks like with 'Einstan' notation.

Let us define X as a l-ranked tensor, $Y = Y(X)$ as a m-ranked tensor, and $Z = Z(Y)$ as a n-ranked tensor. We want to find the $(l+n)$-ranked $\frac{\partial Z}{\partial X}$.

$$\frac{\partial Z}{\partial X} = \frac{\partial Z_{k_1,\ldots,k_n}}{\partial X_{i_1,\ldots,i_l}} = \frac{\partial Z_{k_1,\ldots,k_n}}{\partial Y_{j_1,\ldots,j_m}} \frac{\partial Y_{j_1,\ldots,j_m}}{\partial X_{i_1,\ldots,i_l}}$$

Here our notation says to sum across every unique tuple $(j_1, j_2, \ldots, j_m)$ for every tuple $(k_1, \ldots, k_n, i_1, \ldots, i_l)$.

Let's test this with the following example. Let $W_{i,j} = W_{i,j}(x)$ be a rank-2 tensor, $f = f(W_{i,j})$ be a scalar-valued function. Then,

$$\frac{df}{dW} = \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{\partial f}{\partial W_{i,j}} \frac{dW_{i,j}}{dx} = \frac{\partial f}{\partial W_{i,j}} \frac{dW_{i,j}}{dx}$$

Hopefully, the above example shows how the notation can be understood and applied. And with that, we are now ready to get our hands dirty and tackle the mathematics of neural networks.

6 Finding gradients of neural networks

For simplicity, let's consider a 2-layer neural network. Given

(n, m)-matrix X as the input,

(m, k)-matrix $W^{(1)}$ as layer-1 weights,

(n, k)-matrix $b^{(1)}$ as layer-1 bias,

(k, l)-matrix $W^{(2)}$ as layer-2 weights,

(n, l)-matrix $b^{(2)}$ as layer-2 bias,

σ_1 is the activation function for layer-1,

σ_2 is the activation function for layer-2,

(n, l)-matrix $\hat{Y}$ as the labels,

$E = E(\hat{Y}, Y)$ be the error function

$$Y^{(1)}_{i,j} = X_{i,k} W^{(1)}_{k,j} + b^{(1)}_j$$

$$\hat{Y}^{(1)}_{i,j} = \sigma_1(Y^{(1)})$$

$$Y^{(2)}_{i,j} = \hat{Y}^{(1)}_{i,k} W^{(2)}_{k,j} + b^{(2)}_j$$

$$Y_{i,j} = \sigma_2(Y^{(2)}_{i,j})$$

$$E = E(\hat{Y}, Y)$$

We can see that:

$$\frac{\partial E}{\partial W^{(2)}_{i,j}} = \frac{\partial E}{\partial Y_{a,b}} \frac{\partial Y_{a,b}}{\partial Y^{(2)}_{c,d}} \frac{\partial Y^{(2)}_{c,d}}{\partial W^{(2)}_{i,j}}$$

$$\frac{\partial E}{\partial b^{(2)}_i} = \frac{\partial E}{\partial Y_{a,b}} \frac{\partial Y_{a,b}}{\partial Y^{(2)}_{c,d}} \frac{\partial Y^{(2)}_{c,d}}{\partial b^{(2)}_i}$$

$$\frac{\partial E}{\partial W^{(1)}_{i,j}} = \frac{\partial E}{\partial Y_{a,b}} \frac{\partial Y_{a,b}}{\partial Y^{(2)}_{c,d}} \frac{\partial Y^{(2)}_{c,d}}{\partial \hat{Y}^{(1)}_{e,f}} \frac{\partial \hat{Y}^{(1)}_{e,f}}{\partial Y^{(1)}_{g,h}} \frac{\partial Y^{(1)}_{g,h}}{\partial W^{(1)}_{i,j}}$$

$$\frac{\partial E}{\partial b^{(1)}_i} = \frac{\partial E}{\partial Y_{a,b}} \frac{\partial Y_{a,b}}{\partial Y^{(2)}_{c,d}} \frac{\partial Y^{(2)}_{c,d}}{\partial \hat{Y}^{(1)}_{e,f}} \frac{\partial \hat{Y}^{(1)}_{e,f}}{\partial Y^{(1)}_{g,h}} \frac{\partial Y^{(1)}_{g,h}}{\partial b^{(1)}_i}$$

This is still very complicated but at least we can begin to know what needs to be iterated through to calculate the gradients. In the next article, we will show how we can use these gradients to perform backpropagation and how we can cache gradients to improve efficiency.