Neural Network Gradient Computation

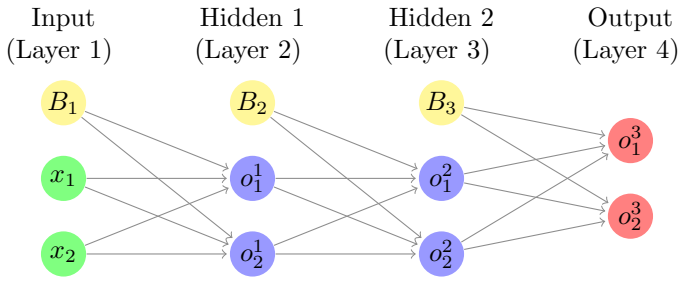
Ansh Vijay

June 2024

Introduction

This document will outline the calculations to determine the gradients for the weight and bias matrices for a 4 layered neural network. More specifically, we will be calculating the gradient of loss with respect to the various weight and bias matrices within the network. The reason we wish to compute these gradients is because they represent how much each individual weight of the network contributed towards the overall loss. Thus, once we compute the gradient we will adjust the weight in the opposite direction in order to minimize loss. Loss is generally how far off the network's output was from the expected output.

The above explanation was solely for a little context. Please note that this is **not** a guide on backpropagation. The network we aim to find gradients for will have 784, 200, 100, and 10 neurons in each layer respectively. Calculating the gradients for layers with this many neurons is a very tedious task. Thus, for this guide we will compute the gradients for the much smaller neural network below, and aim to extrapolate the results to a network with layers of any size.



The gradients computed within this guide are utilized during the backpropagation process, which begins from the output layer and ends at the first hidden layer. However, before beginning to calculate the gradients corresponding to the output layer, I will define some terminology to make the calculations much less verbose.

l – Loss

W_k – Weight matrix connecting layers k and $k+1$

B_k – Bias vector connecting layers k and $k+1$

$w_{i,j}^k$ – Weight connecting neuron i in layer k to neuron j in layer $k+1$

b_j^k – Bias from layer k to neuron j in layer $k+1$

u_k – Input vector to layer k (before applying activation function)

o_k – Output vector of layer k (after applying activation function)

u_i^k – Input to neuron i in layer k (before applying activation function)

o_i^k – Output of neuron i in layer k (after applying activation function)

y_i – Expected output of output neuron i (o_i^3 is the actual output)

Below are the gradients we are trying to compute. The general strategy to compute these gradients will be to first compute $\partial l / \partial w_{i,j}^k$ for all i and j using the chain rule. Then, we will find patterns between each individual gradient to vectorize the gradient computation. In order to calculate individual gradients we will go backwards through the network to arrive at the quantity we wish to compute the gradient of. The reason we go backwards is because loss depends on the network's output, and the output of the network is found by going forwards through each layer. Therefore, if we wish to find out how much each weight/bias contributed to the loss (its gradient), we must begin at the output layer as this is where loss is calculated. However, before heading into computing gradients it will be helpful to understand how the forward pass works.

$$\frac{\partial l}{\partial W_k} \text{ for } k = 1, 2, 3 \qquad \frac{\partial l}{\partial B_k} \text{ for } k = 1, 2, 3$$

$$W_k = \begin{pmatrix} w_{1,1}^k & w_{2,1}^k \\ w_{1,2}^k & w_{2,2}^k \end{pmatrix} \qquad B_k = \begin{pmatrix} b_1^k \\ b_2^k \end{pmatrix}$$

$$\frac{\partial l}{\partial W_k} = \begin{pmatrix} \triangle w_{1,1}^k & \triangle w_{2,1}^k \\ \triangle w_{1,2}^k & \triangle w_{2,2}^k \end{pmatrix} \qquad \frac{\partial l}{\partial B_k} = \begin{pmatrix} \triangle b_1^k \\ \triangle b_2^k \end{pmatrix}$$

Forward Pass

At a high level the forward pass will propagate an input vector through the network by taking a linear combination of the previous layer's outputs o_i^{k-1} and weights $w_{i,j}^k$ for all i in layer k. Next, a bias term b_j^k will be added to the linear combination. Then, an activation function will be applied on this sum to normalize its value within certain bounds. The normalized value will be the input for neuron j in layer $k+1$. Assuming n neurons in layer k and activation function σ, the value of neuron j in layer $k+1$ will take the form:

$$\textbf{(1)}\ \sigma\left(w_{i,j}^k o_i^{k-1} + w_{i+1,j}^k o_{i+1}^{k-1} + ... + w_{n,j}^k o_n^{k-1} + b_j^k\right) = \sigma\left(\sum_{i=1}^{n} w_{i,j}^k o_i^{k-1} + b_j^k\right)$$

This process will continue for all j in layer $k+1$ to form the input vector for layer $k+1$, thereby propagating the original input vector 1 more layer towards the output. Let $u_j^k = \sigma\left(\sum_{i=1}^{n} w_{i,j}^k o_i^{k-1} + b_j^k\right)$ (keep in mind the formula for u_j^k, it will be helpful later on) and assume m neurons in layer $k+1$. Then, after all computation the vector becomes:

$$\textbf{(2)}\ \begin{pmatrix} \sigma\left(u_j^k\right) \\ \sigma\left(u_{j+1}^k\right) \\ \sigma\left(u_{j+2}^k\right) \\ \vdots \\ \sigma\left(u_m^k\right) \end{pmatrix}$$

Computing each u_j^k for each $i, j,$ and k is a very tedious task, which is why matrix multiplication is used for the forward pass. I will go through an example for the first layer, to make it clear why matrix multiplication achieves the same results as above.

Take input vector X, weight matrix W_1, and bias vector B_1. For our network we know the output vector of layer 1 has to be 2×1. Given X is 2×1 and W_1 is 2×2 it is intuitive to matrix multiply them together to form a 2×1 vector. Furthermore, we know a bias term is needed, and it was added after computing the linear combination. Conveniently, adding B_1 to $W_1 \times X$ will maintain a 2×1 shape.

$$(W_1 \times X) + B_1 = \begin{pmatrix} w_{1,1}^1 x_1 + w_{2,1}^1 x_2 + b_1^1 \\ w_{1,2}^1 x_1 + w_{2,2}^1 x_2 + b_2^1 \end{pmatrix}$$

Given that i and $j = 2$, the current vector is similar to **(2)** except that an activation function has not been applied. Because σ is applied element-wise, it will not change the shape of the vector, so we arrive at the formula:

$$o_1 = \sigma\left((W_1 \times X) + B_1\right)$$

This formula can be generalized for all k layers, and is used to take an input vector X and produce an output vector $\hat{y}$.

Output Layer Gradient Calculations

Want: $\frac{\partial l}{\partial W_3}, \frac{\partial l}{\partial B_3}$

(3) $u_1^3 = o_1^2 w_{1,1}^3 + o_2^2 w_{2,1}^3 + b_1^3$

(4) $u_2^3 = o_1^2 w_{1,2}^3 + o_2^2 w_{2,2}^3 + b_2^3$

$$\frac{\partial l}{\partial w_{1,1}^3} = \frac{\partial l}{\partial o_1^3} \cdot \frac{\partial o_1^3}{\partial u_1^3} \cdot \frac{\partial u_1^3}{\partial w_{1,1}^3}$$

$$\frac{\partial l}{\partial w_{1,1}^3} = \frac{\partial l}{\partial u_1^3} \cdot \frac{\partial u_1^3}{\partial w_{1,1}^3}$$

I combined the first 2 terms because $\partial l / \partial u_1^3$ is a convenient term to calculate. Since Softmax is used as the activation function for the output layer, this value ends up becoming $y_1 - \hat{y}_1$. To see the steps of computing Softmax's derivative see this article. Additionally, according to **(3)** we can clearly see $\partial u_1^3 / \partial w_{1,1}^3 = o_1^2$. Thus,

$$\frac{\partial l}{\partial w_{1,1}^3} = \left(y_1 - o_1^3\right) \cdot o_1^2$$

Following, the same process for the rest of the weights in W_3 we see:

$$\frac{\partial l}{\partial w_{1,1}^3} = \left(y_1 - o_1^3\right) \cdot o_1^2$$

$$\frac{\partial l}{\partial w_{1,2}^3} = \left(y_2 - o_2^3\right) \cdot o_1^2$$

$$\frac{\partial l}{\partial w_{2,1}^3} = \left(y_1 - o_1^3\right) \cdot o_2^2$$

$$\frac{\partial l}{\partial w_{2,2}^3} = \left(y_2 - o_2^3\right) \cdot o_2^2$$

$$\frac{\partial l}{\partial W_3} = \begin{pmatrix} \left(y_1 - o_1^3\right) \cdot o_1^2 & \left(y_1 - o_1^3\right) \cdot o_2^2 \\ \left(y_2 - o_2^3\right) \cdot o_1^2 & \left(y_2 - o_2^3\right) \cdot o_2^2 \end{pmatrix}$$

$$\frac{\partial l}{\partial W_3} = (y - o_3) \times o_2^T$$

To compute the gradients for B_3 we can notice that the gradient will be identical to the gradients of the individual weights, except instead of $\partial u_1^3 / \partial w_{i,j}^k$ the second term will be $\partial u_1^3 / \partial b_j^k$, which according to **(3)** and **(4)**, is 1.

$$\frac{\partial l}{\partial W_3} = (y - o_3) \times o_2^T \qquad \frac{\partial l}{\partial B_3} = (y - o_3)$$

Hidden Layer 2 Gradient Calculations

Want: $\frac{\partial l}{\partial W_2}, \frac{\partial l}{\partial B_2}$

(5) $u_1^2 = o_1^1 w_{1,1}^2 + o_2^1 w_{2,1}^2 + b_1^2$

(6) $u_2^2 = o_1^1 w_{1,2}^2 + o_2^1 w_{2,2}^2 + b_2^2$

From the output neurons, there is more than one way to arrive at $w_{i,j}^k$. To account for this in gradient computation, all "paths" are added together since they all contribute to the loss. For this layer, each gradient will have 2 terms. For example, for $w_{1,1}^2$ the two "paths" are:

$$l \to o_1^3 \to u_1^3 \to o_1^2 \to u_1^2 \to w_{1,1}^2$$

$$l \to o_2^3 \to u_2^3 \to o_1^2 \to u_1^2 \to w_{1,1}^2$$

$$\frac{\partial l}{\partial w_{1,1}^2} = \left(\frac{\partial l}{\partial o_1^3} \cdot \frac{\partial o_1^3}{\partial u_1^3} \cdot \frac{\partial u_1^3}{\partial o_1^2} \cdot \frac{\partial o_1^2}{\partial u_1^2} \cdot \frac{\partial u_1^2}{\partial w_{1,1}^2}\right) + \left(\frac{\partial l}{\partial o_2^3} \cdot \frac{\partial o_2^3}{\partial u_2^3} \cdot \frac{\partial u_2^3}{\partial o_1^2} \cdot \frac{\partial o_1^2}{\partial u_1^2} \cdot \frac{\partial u_1^2}{\partial w_{1,1}^2}\right)$$

$$\frac{\partial l}{\partial w_{1,1}^2} = \left(\frac{\partial l}{\partial u_1^3} \cdot \frac{\partial u_1^3}{\partial o_1^2} \cdot \frac{\partial o_1^2}{\partial u_1^2} \cdot \frac{\partial u_1^2}{\partial w_{1,1}^2}\right) + \left(\frac{\partial l}{\partial u_2^3} \cdot \frac{\partial u_2^3}{\partial o_1^2} \cdot \frac{\partial o_1^2}{\partial u_1^2} \cdot \frac{\partial u_1^2}{\partial w_{1,1}^2}\right)$$

Note: $\sigma'()$ represents the derivative of the activation function used when computing o_1^2. Thus, in order to compute $\partial o_1^2 / \partial u_1^2$ we see that

$$\sigma(u_1^2) = o_1^2$$

$$\frac{\partial o_1^2}{\partial u_1^2} = \frac{\partial \left(\sigma(u_1^2)\right)}{\partial u_1^2} = \sigma'(u_1^2)$$

$$\frac{\partial l}{\partial w_{1,1}^2} = \left((y_1 - o_1^3) \cdot w_{1,1}^3 \cdot \sigma'(u_1^2) \cdot o_1^1\right) + \left((y_2 - o_2^3) \cdot w_{1,2}^3 \cdot \sigma'(u_1^2) \cdot o_1^1\right)$$

$$\frac{\partial l}{\partial w_{1,1}^2} = \sigma'(u_1^2)(o_1^1)\left((y_1 - o_1^3)(w_{1,1}^3) + (y_2 - o_2^3)(w_{1,2}^3)\right)$$

The other weights in W_2 end up being:

$$\frac{\partial l}{\partial w_{1,1}^2} = \left(\sigma'(u_1^2)\right)\left(o_1^1\right)\left((y_1 - o_1^3)(w_{1,1}^3) + (y_2 - o_2^3)(w_{1,2}^3)\right)$$

$$\frac{\partial l}{\partial w_{1,2}^2} = \left(\sigma'(u_2^2)\right)\left(o_1^1\right)\left((y_1 - o_1^3)(w_{2,1}^3) + (y_2 - o_2^3)(w_{2,2}^3)\right)$$

$$\frac{\partial l}{\partial w^2_{2,1}} = \left(\sigma'(u^2_1)\right)\left(o^1_2\right)\left((y_1 - o^3_1)(w^3_{1,1}) + (y_2 - o^3_2)(w^3_{1,2})\right)$$

$$\frac{\partial l}{\partial w^2_{2,2}} = \left(\sigma'(u^2_2)\right)\left(o^1_2\right)\left((y_1 - o^3_1)(w^3_{2,1}) + (y_2 - o^3_2)(w^3_{2,2})\right)$$

The factoring we did after computing the gradients allows us to see a general pattern $(A)(B)(CD + EF)$. From the example done in the Forward Pass section we saw that matrix multiplication helped us compute linear combinations. Inspecting the $CD + EF$ portion of each gradient we notice two similar quantities: weights from W_3 and the same $(y - o_3)$ vector from the previous gradient. We can also notice this $CD + EF$ quantity only has 2 variations as the linear combinations are the same in the gradients for $w^2_{1,1}, w^2_{2,1}$ and $w^2_{1,2}, w^2_{2,2}$, thus using W_3 and $(y - o_3)$ we should aim to produce a 2×1 matrix. The only two multiplications that could satisfy these conditions would be $W_3 \times (y - o_3)$ or $W^T_3 \times (y - o_3)$. Ultimately, $W^T_3 \times (y - o_3)$ is the correct choice, since one of the values in the $W_3 \times (y - o_3)$ matrix is $\left(y_1 - o^3_1\right)\left(w^3_{1,1}\right) + \left(y_2 - o^3_2\right)\left(w^3_{2,1}\right)$, which does not exist in any of the gradients above.

Now, we must address the A and B terms, which are $\sigma'(u_2)$ (2×1) and o_1 (2×1) respectively. In order to see how we should use these terms to construct $\partial l / \partial W_2$ it is helpful to see what we have so far, which we'll call $\partial l / \partial W_{2a}$, and what $\partial l / \partial W_2$ should be according to the calculations of the individual gradients.

$$\text{Let } d_1 = (y_1 - o^3_1) \text{ and } d_2 = (y_2 - o^3_2)$$

$$\frac{\partial l}{\partial W_{2a}} = \begin{pmatrix} d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2} \\ d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2} \end{pmatrix} = W^T_3 \times (y - o_3)$$

$$\frac{\partial l}{\partial W_2} = \begin{pmatrix} \left(\sigma'(u^2_1)\right)\left(o^1_1\right)\left(d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2}\right) & \left(\sigma'(u^2_1)\right)\left(o^1_2\right)\left(d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2}\right) \\ \left(\sigma'(u^2_2)\right)\left(o^1_1\right)\left(d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2}\right) & \left(\sigma'(u^2_2)\right)\left(o^1_2\right)\left(d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2}\right) \end{pmatrix}$$

The most obvious difference between $\partial l / \partial W_{2a}$ and $\partial l / \partial W_2$ is the difference in shape. In order to achieve the 2×2 shape we must matrix multiply $\partial l / \partial W_{2a}$ by a 1×2 vector. This can be achieved by taking the transpose of either o_1 or $\sigma'(u_2)$. By inspection we see o^T_1 will pair the correct terms together.

$$\frac{\partial l}{\partial W_{2b}} = \begin{pmatrix} d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2} \\ d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2} \end{pmatrix} \times \begin{pmatrix} o^1_1 & o^1_2 \end{pmatrix}$$

$$\frac{\partial l}{\partial W_{2b}} = \begin{pmatrix} \left(o^1_1\right)\left(d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2}\right) & \left(o^1_2\right)\left(d_1 \cdot w^3_{1,1} + d_2 \cdot w^3_{1,2}\right) \\ \left(o^1_1\right)\left(d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2}\right) & \left(o^1_2\right)\left(d_1 \cdot w^3_{2,1} + d_2 \cdot w^3_{2,2}\right) \end{pmatrix}$$

Finally, we must element-wise multiply $(\odot)$ $\sigma'(u_2)$ with $\partial l / \partial W_{2b}$. However, because of the difference in shape the equation becomes:

$$\frac{\partial l}{\partial W_2} = \frac{\partial l}{\partial W_{2b}} \odot \begin{pmatrix} \sigma'(u_2) & \sigma'(u_2) \end{pmatrix}$$

This equation is awkward because of the concatenation of the $\sigma'(u_2)$ vectors, and will be hard to deal with when used for the gradient calculations for W_1 and B_1. To avoid this concatenation, we can instead first element-wise multiply $\sigma'(u_2)$ with $\partial l/\partial W_{2a}$ since they have the same shape, and then matrix multiply by o_1^T.

$$\frac{\partial l}{\partial W_2} = \left(\frac{\partial l}{\partial W_{2a}} \odot \sigma'(u_2)\right) \times o_1^T$$

$$\frac{\partial l}{\partial W_2} = \left(\left(W_3^T \times (y - o_3)\right) \odot \sigma'(u_2)\right) \times o_1^T$$

Just like in the output layer, for calculating $\partial l/\partial B_2$ we notice the equations for the gradients for individual biases are the same as the individual weights, except the last term becomes $\partial u_i^2/\partial b_j^2$ instead of $\partial u_i^2/\partial w_{i,j}^2$. From **(5)** and **(6)** we see $\partial u_i^2/\partial b_j^2$ becomes 1 regardless of i or j. Thus, for $\partial l/\partial B_2$ we simply don't matrix multiply by o_1^T, which makes sense because this achieves a (2×1) shape.

$$\frac{\partial l}{\partial W_2} = \left(\left(W_3^T \times (y - o_3)\right) \odot \sigma'(u_2)\right) \times o_1^T \qquad \frac{\partial l}{\partial B_2} = \left(\left(W_3^T \times (y - o_3)\right) \odot \sigma'(u_2)\right)$$

Hidden Layer 1 Gradient Calculations

Want: $\frac{\partial l}{\partial W_1}, \frac{\partial l}{\partial B_1}$

(7) $u_1^1 = x_1 w_{1,1}^1 + x_2 w_{2,1}^1 + b_1^1$

(8) $u_2^1 = x_1 w_{1,2}^1 + x_2 w_{2,2}^1 + b_2^1$

Now that we are one layer deeper, each individual weight/bias gradient will have four terms because there are four "paths" to each quantity. Hopefully, one can see that for a neural network with two neurons in each layer the number of ways to each weight or bias is equal to 2^{n-k}, where n is the number of layers and k is the layer we are computing the gradients for.

With four terms in each gradient the computation becomes very long and repetitive, so instead of computing all four weight gradients we will compute only the gradient for $w_{1,1}^1$, and then discover a pattern in the transition from $\partial l/\partial W_3$ to $\partial l/\partial W_2$, which we will use to vectorize $\partial l/\partial W_1$.

$$\frac{\partial l}{\partial w_{1,1}^1} = \left(\frac{\partial l}{\partial u_1^3}\cdot\frac{\partial u_1^3}{\partial o_1^2}\cdot\frac{\partial o_1^2}{\partial u_1^2}\cdot\frac{\partial u_1^2}{\partial o_1^1}\cdot\frac{\partial o_1^1}{\partial u_1^1}\cdot\frac{\partial u_1^1}{\partial w_{1,1}^1}\right)+\left(\frac{\partial l}{\partial u_1^3}\cdot\frac{\partial u_1^3}{\partial o_2^2}\cdot\frac{\partial o_2^2}{\partial u_2^2}\cdot\frac{\partial u_2^2}{\partial o_1^1}\cdot\frac{\partial o_1^1}{\partial u_1^1}\cdot\frac{\partial u_1^1}{\partial w_{1,1}^1}\right)+$$

$$\left(\frac{\partial l}{\partial u_2^3}\cdot\frac{\partial u_2^3}{\partial o_1^2}\cdot\frac{\partial o_1^2}{\partial u_1^2}\cdot\frac{\partial u_1^2}{\partial o_1^1}\cdot\frac{\partial o_1^1}{\partial u_1^1}\cdot\frac{\partial u_1^1}{\partial w_{1,1}^1}\right)+\left(\frac{\partial l}{\partial u_2^3}\cdot\frac{\partial u_2^3}{\partial o_2^2}\cdot\frac{\partial o_2^2}{\partial u_2^2}\cdot\frac{\partial u_2^2}{\partial o_1^1}\cdot\frac{\partial o_1^1}{\partial u_1^1}\cdot\frac{\partial u_1^1}{\partial w_{1,1}^1}\right)$$

$$\begin{aligned}\frac{\partial l}{\partial w_{1,1}^1} = &\left(\left(y_1 - o_1^3\right)\left(w_{1,1}^3\right)\left(\sigma'\left(u_1^2\right)\right)\left(w_{1,1}^2\right)\left(\sigma'\left(u_1^1\right)\right)\left(x_1\right)\right)+\\ &\left(\left(y_1 - o_1^3\right)\left(w_{2,1}^3\right)\left(\sigma'\left(u_2^2\right)\right)\left(w_{1,2}^2\right)\left(\sigma'\left(u_1^1\right)\right)\left(x_1\right)\right)+\\ &\left(\left(y_2 - o_2^3\right)\left(w_{1,2}^3\right)\left(\sigma'\left(u_1^2\right)\right)\left(w_{1,1}^2\right)\left(\sigma'\left(u_1^1\right)\right)\left(x_1\right)\right)+\\ &\left(\left(y_2 - o_2^3\right)\left(w_{2,2}^3\right)\left(\sigma'\left(u_2^2\right)\right)\left(w_{1,2}^2\right)\left(\sigma'\left(u_1^1\right)\right)\left(x_1\right)\right)+\end{aligned}$$

Let $d_1 = (y_1 - o_1^3)$ and $d_2 = (y_2 - o_2^3)$

$$\begin{aligned}\frac{\partial l}{\partial w_{1,1}^1} = &\left(\sigma'\left(u_1^1\right)\cdot x_1\right)\left(\left(\sigma'\left(u_1^2\right)\cdot w_{1,1}^2\right)\left(d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\right)+\right.\\ &\left.\left(\sigma'\left(u_2^2\right)\cdot w_{1,2}^2\right)\left(d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\right)\right)\end{aligned}$$

We see that $\partial l/\partial W_3$ is formed by multiplying the error with the transpose of the outputs from the previous layer:

$$(y - o_3)\times o_2^T$$

In $\partial l/\partial W_2$ we once again see the error and the transpose of the outputs from the previous layer (o_1^T):

$$\frac{\partial l}{\partial W_2} = \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big) \times o_1^T$$

Additionally, it is useful to enumerate the steps we took to compute $\partial l/\partial W_2$.

(9) Matrix multiply the propagated error by the transpose of the weight matrix of the next layer $\big(W_3^T\big)$

(10) Element-wise multiply this product by the derivative of the activation function on the current layer's inputs $(\sigma'(u_2))$

(11) Matrix multiply by the transpose of the previous layer's output $\big(o_1^T\big)$.

To construct $\partial l/\partial W_1$ we will do the above steps in reverse order and see if we end up with a formula that would in fact compute our current value for $\partial l/\partial w_{1,1}^1$. Before beginning the calculations, it will be helpful to keep in mind that the gradient for $w_{1,1}^1$ will be the top left entry in $\partial l/\partial W_1$

$$\textbf{(12)} \quad \frac{\partial l}{\partial W_1} = \begin{pmatrix} \triangle w_{1,1}^1 & \triangle w_{2,1}^1 \\ \triangle w_{1,2}^1 & \triangle w_{2,2}^1 \end{pmatrix}$$

First, we address **(11)**, which means we will matrix multiply some (2×1) vector by X^T to form the 2×2 $\partial l/\partial W_1$ matrix:

$$\textbf{(13)} \quad \frac{\partial l}{\partial W_1} = \begin{pmatrix} A_0 \\ B_0 \end{pmatrix} \times X^T = \begin{pmatrix} A_0 x_1 & A_0 x_2 \\ B_0 x_1 & B_0 x_2 \end{pmatrix}$$

Next, according to **(10)** we will element-wise multiply by $\sigma'(u_1)$, which means:

$$\begin{pmatrix} A_0 \\ B_0 \end{pmatrix} = \begin{pmatrix} A_1 \\ B_1 \end{pmatrix} \odot \sigma'(u_1) = \begin{pmatrix} A_1 \cdot \sigma'(u_1^1) \\ B_1 \cdot \sigma'(u_2^1) \end{pmatrix}$$

(9) implies that we will matrix multiply by W_2^T:

$$\begin{pmatrix} A_1 \\ B_1 \end{pmatrix} = W_2^T \times \begin{pmatrix} A_2 \\ B_2 \end{pmatrix} = \begin{pmatrix} A_2 w_{1,1}^2 + B_2 w_{1,2}^2 \\ A_2 w_{2,1}^2 + B_2 w_{2,2}^2 \end{pmatrix}$$

Because of **(12)** and **(13)** we know $\partial l/\partial w_{1,1}^1 = A_0 x_1$. Thus, it would be helpful to write A_0 in terms of A_2 and B_2 in order to find their values.

$$\begin{pmatrix} A_0 \\ B_0 \end{pmatrix} = \begin{pmatrix} A_1 \cdot \sigma'(u_1^1) \\ B_1 \cdot \sigma'(u_2^1) \end{pmatrix}$$

$$\begin{pmatrix} A_0 \\ B_0 \end{pmatrix} = \begin{pmatrix} \big(A_2 w_{1,1}^2 + B_2 w_{1,2}^2\big) \cdot \sigma'(u_1^1) \\ \big(A_2 w_{2,1}^2 + B_2 w_{2,2}^2\big) \cdot \sigma'(u_2^1) \end{pmatrix}$$

$$\frac{\partial l}{\partial w_{1,1}^1} = A_0 x_1 = \left(A_2 w_{1,1}^2 + B_2 w_{1,2}^2\right)\left(\sigma'\left(u_1^1\right)\cdot x_1\right)$$

$$\begin{aligned}\left(A_2 w_{1,1}^2 + B_2 w_{1,2}^2\right)\left(\sigma'\left(u_1^1\right)\cdot x_1\right) = \left(\sigma'\left(u_1^1\right)\cdot x_1\right)\Big(&\left(\sigma'\left(u_1^2\right)\cdot w_{1,1}^2\right)\left(d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\right) + \\ &\left(\sigma'\left(u_2^2\right)\cdot w_{1,2}^2\right)\left(d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\right)\Big)\end{aligned}$$

$$\begin{aligned}A_2 w_{1,1}^2 + B_2 w_{1,2}^2 = &\left(\sigma'\left(u_1^2\right)\cdot w_{1,1}^2\right)\left(d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\right) + \\ &\left(\sigma'\left(u_2^2\right)\cdot w_{1,2}^2\right)\left(d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\right)\end{aligned}$$

$$\begin{aligned}A_2 w_{1,1}^2 &= \left(\sigma'\left(u_1^2\right)\cdot w_{1,1}^2\right)\left(d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\right)\\ B_2 w_{1,2}^2 &= \left(\sigma'\left(u_2^2\right)\cdot w_{1,2}^2\right)\left(d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\right)\end{aligned}$$

$$\begin{pmatrix}A_2\\B_2\end{pmatrix} = \begin{pmatrix}\sigma'\left(u_1^2\right)\cdot\left(d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\right)\\ \sigma'\left(u_2^2\right)\cdot\left(d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\right)\end{pmatrix}$$

Now that we've computed the values for A_2 and B_2 we see some familiar linear combinations, which we have vectorized during the hidden layer 2 gradient calculations (see $\partial l/\partial W_{2a}$).

$$\begin{pmatrix}A_2\\B_2\end{pmatrix} = \begin{pmatrix}A_3\\B_3\end{pmatrix}\odot\sigma'(u_2)$$

$$\begin{pmatrix}A_3\\B_3\end{pmatrix} = \begin{pmatrix}d_1\cdot w_{1,1}^3 + d_2\cdot w_{1,2}^3\\ d_1\cdot w_{2,1}^3 + d_2\cdot w_{2,2}^3\end{pmatrix} = W_3^T\times(y-o_3)$$

Finally, we can keep replacing values of $\begin{pmatrix}A_n\\B_n\end{pmatrix}$ into $\begin{pmatrix}A_{n-1}\\B_{n-1}\end{pmatrix}$ to arrive at a formula for $\partial l/\partial W_1$

$$\begin{pmatrix}A_2\\B_2\end{pmatrix} = \left(W_3^T\times(y-o_3)\right)\odot\sigma'(u_2)$$

$$\begin{pmatrix}A_1\\B_1\end{pmatrix} = W_2^T\times\left(\left(W_3^T\times(y-o_3)\right)\odot\sigma'(u_2)\right)$$

$$\begin{pmatrix}A_0\\B_0\end{pmatrix} = \left(W_2^T\times\left(\left(W_3^T\times(y-o_3)\right)\odot\sigma'(u_2)\right)\right)\odot\sigma'(u_1)$$

$$\frac{\partial l}{\partial W_1} = \left(\left(W_2^T\times\left(\left(W_3^T\times(y-o_3)\right)\odot\sigma'(u_2)\right)\right)\odot\sigma'(u_1)\right)\times X^T$$

For $\partial l/\partial B_1$ we can notice once again that the last term in each "path" would be $\partial u_i^1/\partial b_j^1$ which according to **(7)** and **(8)** is 1, thus there would be no x_1 or x_2 terms.

$$\frac{\partial l}{\partial W_1} = \Big(\big(W_2^T \times \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big)\big) \odot \sigma'(u_1)\Big) \times X^T$$

$$\frac{\partial l}{\partial B_1} = \Big(\big(W_2^T \times \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big)\big) \odot \sigma'(u_1)\Big)$$

Extensibility

$$\frac{\partial l}{\partial W_3} = (y - o_3) \times o_2^T \quad \frac{\partial l}{\partial B_3} = (y - o_3)$$

$$\frac{\partial l}{\partial W_2} = \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big) \times o_1^T$$

$$\frac{\partial l}{\partial B_2} = \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big)$$

$$\frac{\partial l}{\partial W_1} = \big(\big(W_2^T \times \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big)\big) \odot \sigma'(u_1)\big) \times X^T$$

$$\frac{\partial l}{\partial B_1} = \big(\big(W_2^T \times \big(\big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)\big)\big) \odot \sigma'(u_1)\big)$$

Above are the gradients we have calculated for all layers and all biases within our four layered neural network. By inspection, one can see the overlap between expressions for $\partial l/\partial W_k, \partial l/\partial W_{k-1}$ and $\partial l/\partial B_k, \partial l/\partial B_{k-1}$. This overlap is the error that is propagated from the output layer all the way to the input layer during backpropagation, which motivates weight changes in a direction that minimizes loss. (For a proper explanation of backpropagation consider a trusted source.)

In our case the error starts out as $(y - o_3)$, which we see again in $\partial l/\partial W_2$. As outlined by **(9)**, **(10)**, and **(11)**, this error is operated on in order to produce $\partial l/\partial W_2$. Going from the gradient calculations for W_2 to W_1 we see that most of $\partial l/\partial W_2$ was propagated to $\partial l/\partial W_1$, except for **(11)**. This was the same case for $\partial l/\partial W_3 \rightarrow \partial l/\partial W_2$. Thus, it is fair to assume the error propagated from layer 3 to 2 is $\delta^2 = \big(W_3^T \times (y - o_3)\big) \odot \sigma'(u_2)$. This assumption is further supported because just like for $\partial l/\partial W_2$, $\partial l/\partial W_1$ was computed by performing **(9)**, **(10)**, and **(11)** on δ^2.

We can start to see a pattern where we can compute the current weight's gradient by performing **(9)**, **(10)**, and **(11)** on the previous layer's error. Additionally, the previous layer's error simply becomes the gradient for the current layer's bias vector. Thus, we can extend gradient computations to neural networks for n layers using the following equations.

$$\text{Output Layer Error } \Rightarrow \delta^n = (y - o_n)$$

$$\text{Layer } k \text{ Error } \Rightarrow \delta^k = \big(W_{k+1} \times \delta^{k+1}\big) \odot \sigma'(u_k)$$

$$\text{Weight Gradient } \Rightarrow \frac{\partial l}{\partial W_k} = \delta^k \times o_{k-1}^T$$

$$\text{Bias Gradient } \Rightarrow \frac{\partial l}{\partial B_k} = \delta^k$$