How I think about the neural network backpropagation algorithm

Carolyn Johnston

January 6, 2020

1 Introduction

This writeup owes a lot to Chapter 2 in Michael Neilsen's free online book, "Neural Networks and Deep Learning" [1]; I first got my arms around the mathematics of the backpropagation algorithm by studying that chapter very carefully. Another really delightful treatment of the mathematics behind the backpropagation algorithm is 3Blue1Brown's video of the calculus of backpropagation ([3]), which explains everything very clearly for the case where all the neural network layers are 1-dimensional.

At the heart of the neural net backpropagation algorithm is a clever algorithm for computing the gradient of the carefully constructed neural net cost function. This gradient is needed in order to train the neural network using optimization algorithms based on gradient descent methods.

Since neural network functions are composite functions, the chain rule figures very large in the calculation of the gradient. Viewed during the post-training operation of the neural network, with all the weight and bias parameters fixed, the neural network is a function of its inputs; but during training, the neural network must be thought of as having fixed input/output pairs, and being a function of its parameters. As part of training, the gradient must be computed with respect to all of its potentially millions of parameters. This is the central task of the backpropagation algorithm.

What makes understanding the neural network backpropagation algorithm so challenging? I think it is trying to understand the big picture, while simultaneously tracking indices and summations for chain-rule expressions involving high-dimensional function values and arguments. Our minds just aren't equipped to do both things at the same time. This is why matrix notation for linear algebra is such a game-changer; in our undergraduate linear algebra classes, we learned to 'chunk' the meanings of matrix expressions, and everything about linear algebra since then has been easier to understand.

In this writeup, I show that you can derive the standard high-dimensional backpropagation equations by thinking of the terms in the chain rule as being vectors, matrices, and in some cases, 3-dimensional 'boxes of numbers', with appropriate definitions for their products. The vectors, matrices and 3-dimensional boxes of numbers are actually 1-, 2-, and 3-tensors with appropriate tensor products; tensor algebra is at the core of neural net algorithms, but most of us aren't comfortable with tensor algebra.

My goal here isn't to teach tensor algebra, but rather to show that the chain rule, expressed in terms of these objects, is as easy to understand as the one-dimensional chain rule case; but the multidimensional algorithm is still derivable in its fullest high-dimensional complexity.

2 The feed-forward neural network and the cost function

Figure 1 shows a high-level schematic of a simple neural network with an input layer 1 of size 3, a hidden layer 2 of size 4, and an output layer 3 of size 2 (let's ignore the '1' nodes for the moment). This simple neural network computes a function $f : \mathbb{R}^3 \to \mathbb{R}^2$. In order to train this neural net, we would need a (large) set of sample input - output pairs

$$D = \{(\vec{z}_i, \lambda_i)\}_{i=1}^N,$$

where $\vec{z}_i \in \mathbb{R}^3$ and $\lambda_i = g(\vec{z}_i) \in \mathbb{R}^2$ are sample inputs and outputs from the neural network. Our goal is to use the sample data to teach the neural net to compute $g(z)$ for examples of z that are not in the training set.

In the neural network graph in Figure 1, each node in layers 2 and 3 with an edge pointing to it (reading left to right) represents a point in the computation at which two things happen. First, all the inputs to the node are summed; second, a nonlinear activation function is applied. Each edge in the graph is associated with a real-valued weight w that is applied, via multiplication, to the output of the previous node before it is input to the new node.

In order to train the neural network using gradient descent, we need to have a scalar-valued function $\mathbb{E}$ to take the gradient of. $\mathbb{E}$ can be thought of as measuring the error in the output of the neural net, and the goal of gradient descent is to tune the parameters of the neural net until this error is minimized.

Usually, $\mathbb{E}$ is the sum or average over all the sample input-output pairs of a cost function, $C(f(\vec{z}_i), \lambda_i)$, that expresses the cost of obtaining the output $f(\vec{z}_i)$ when the correct output is $\lambda_i = g(\vec{z}_i)$.

If $f(\vec{z}_i) \equiv g(\vec{z}_i) = \lambda_i$, the cost function $C(f(\vec{z}_i), \lambda_i)$ is typically 0, indicating that there is no cost if the answer given by the neural net is correct. The cost function takes on positive values when $f(\vec{z}_i) \neq \lambda_i$. The objective of training the network is to tune the weights associated with the edges in the neural net so that the total error $\mathbb{E} = \Sigma_i C(f(\vec{z}_i), \lambda_i)$ is minimized. One of the simplest examples of a cost function is the squared difference between the values, $C(f(\vec{z}_i), \lambda_i) = (f(\vec{z}_i) - \lambda_i)^2$.

Figure 2 shows a more detailed schematic of the transforms in a single layer. In particular, it shows how the input vectors $\vec{z}_i^{l-1}$ to nodes in layer $l-1$ are transformed to create the inputs $\vec{z}_i^l$ to layer l. In Figure 2, I have separated

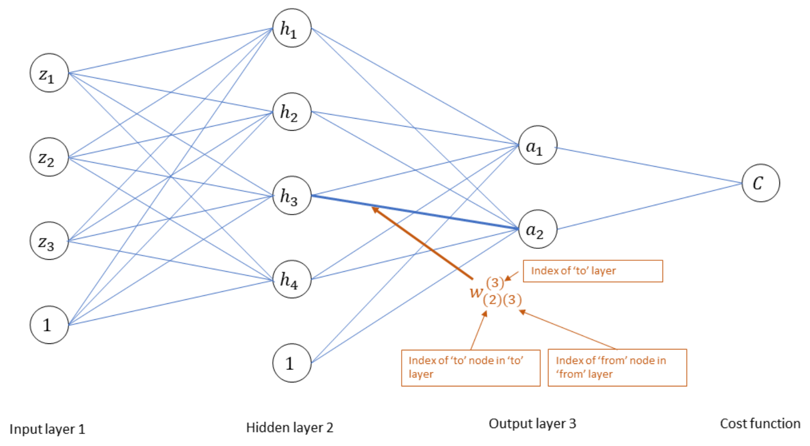

Figure 1: Schematic of a neural network, showing standard weight labeling scheme.

each node in the graph in Figure 1 into two nodes, each of which applies a single operation to its input. The summation nodes (shown with a $+$ sign) sum all their inputs. The σ nodes apply the activation functions σ. In most neural net graphics, these two operations are represented together by a single node, as they were in Figure 1.

2.1 The scalar viewpoint

We'll first look at the feed-forward algorithm in terms of individual vector components, rather than vectors. Figure 2 shows each scalar component z_i^{l-1} of an input vector individually. The scalar z_i^{l-1} is first passed through a σ node, which applies a nonlinear 'activation' function to it. Classically, the activation function was chosen to be the logistic function

$$\sigma(x) = \frac{1}{1+\exp(-x)},$$

but there are other activation functions in wide use. In particular, the ReLU activation function

$$\sigma(x) = \max(0, x)$$

is widely used in convolutional neural nets to solve computer vision problems.

In Figure 2, both the σ nodes and their scalar outputs, $\sigma(z_i^{l-1}) = a_i^{l-1}$, are shown in green. Each output value a_i^{l-1} from the i-th σ node is then transmitted to the sum $(+)$ nodes (shown in blue) in layer $l-1$ through all of the edges/connectors that originate at the i-th σ node. If an edge connects the j-th sigma node in layer $l-1$ to the i-th sum node, then the value a_j^{l-1} will be multiplied by the weight w_{ij}^l during transmission. Both the weights and their associated edges are shown in orange.

Note also that there are special nodes in layer $l-1$, labeled with a '1'; these only emit 1s, and do not depend on any input from the previous layer of the neural net. The edge which connects such a node to the i-th summation node multiplies the emitted 1 by a bias parameter, b_i^l.

2.2 The vector viewpoint

Since the weights w_{ij}^l are doubly indexed, we can think of these weights as constituting a matrix which we call W^l. The matrix W^l transforms the vector $\vec{a}_j^{l-1}$ by matrix multiplication before it reaches the next summation step. The dimensions of the matrix W^l are $N_l \times N_{l-1}$, where N_l is the number of sum nodes in the layer that that the edges are pointing to, and N_{l-1} is the number of σ nodes that the edges originate from.

Similarly, the parameters b_i^l constitute a vector of length N_l, which we call $\vec{b^l}$. In Figure 2, $N_l = 2$ and $N_{l-1} = 3$, so W^l is a 2x3 matrix, and $\vec{b^l}$ is a 2-vector.

For a fully connected neural network, there are a total of $N_{l-1}+1$ edges that arrive at each summation node in layer $l-1$. N_{l-1} of them carry the values $a_j^{l-1} * w_{ij}^l$ for $j = 1, ..., N_l$, and one carries the value b_i^l coming from the '1' node. These are summed at the summation node to give

$$z_i^l = \Sigma_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l.$$

However, this summation operation can be written more succinctly using the matrix product:

$$\vec{z^l} = W^l \cdot \vec{a}^{l-1} + \vec{b}^l,$$

where $\vec{z^l}, \vec{b^l} \in \mathbb{R}^{N_l}$, and $\vec{a}^{l-1} \in \mathbb{R}^{N_{l-1}}$.

If we define the vectorized form of the scalar function σ as

$$\sigma(\vec{x}) = (\sigma(x_1), ..., \sigma(x_n))^t,$$

and suppress the vector notation and the matrix indices, the function ϕ^l that transforms $\vec{z^l}^{-1}$ to $\vec{z^l}$ is given by

$$\vec{z^l} = \phi^l(\vec{z^l}^{-1}) = W^l \cdot \sigma(\vec{z^l}^{-1}) + b^l. \quad (1)$$

The neural network function f is then defined as

$$f(\vec{z^1}) = \sigma(\phi^L(\phi^{L-1}(....(\phi^2(\vec{z^1})...)) = \sigma(\phi^L \circ \phi^{L-1} \circ ... \circ \phi^2(\vec{z^1})). \quad (2)$$

The game is to tune the weight parameters in W^l and the bias parameters in $\vec{b^l}$, for each layer l, so that the function $f(z)$ is close to the desired function $g(z)$.

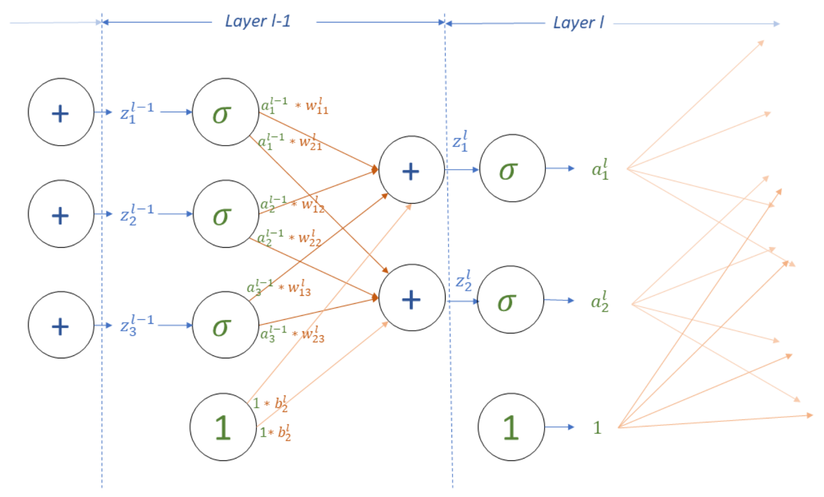

Figure 2: Schematic of a single layer of a neural network.

3 Some linear algebra and multivariate calculus notation

It is possible to define derivatives of scalar-, vector-, and matrix-valued functions with respect to arguments that are scalar, vector, or matrix valued; and it is possible to do so in such a way that the chain rule can be neatly expressed in terms of products of these objects.

The gradient and the Jacobian are examples of higher-dimensional derivative objects, and there are other extensions. "The Matrix Cookbook" ([4]) is a popular online reference that discusses derivatives of scalar functions with respect to a matrix argument. "Old and New Matrix Algebra Useful for Statistics" ([5]) is a reference which discusses all the types of derivatives that can be expressed as matrices. Neither of these references discusses derivatives that must be expressed as higher-dimensional objects than matrices.

In the sections below, I talk about what sort of objects these higher-dimensional derivative objects are, and how they can be multiplied together to express the chain rule in various situations. My approach to this uses a sort of 'dimensional analysis', answering the question: if the standard expressions of calculus, such as the Taylor expansion and the chain rule, are to make sense in each case, what kind of objects do their elements need to be, how are they multiplied, and what are their dimensions?

I am assuming that, like me, you are very familiar with matrix multiplication, and much less familiar with tensor multiplication. A reference I liked for this topic is Chapter 1 in “A Gentle Introduction to Tensors”, by Boaz Porat ([2]).

We’ll assume that every vector, whether it is the value of a function or the argument of a function, is a column vector (equivalently, a matrix of dimension $N \times 1$).

3.1 Derivative of a vector-valued function of a scalar

Suppose $f : \mathbb{R} \to \mathbb{R}^{N_l}$ is a N_l-dimensional vector-valued function of a scalar, z. Then we should have $\vec{f}(z + \Delta z) \approx \vec{f}(z) + [\delta\vec{f}/\delta z] \cdot \Delta z$, and so $[\delta\vec{f}/\delta z] \cdot \Delta z$ must also be a N_l-dimensional column vector. Since Δz is a scalar, $[\delta\vec{f}/\delta z]$ must be an N_l-dimensional vector, and the two are multiplied using the standard scalar product.

3.2 Derivative of a scalar-valued function of a vector

If $f : \mathbb{R}^{N_l} \to \mathbb{R}$ is a scalar-valued function of a vector, so that $f(\vec{z}^l + \Delta\vec{z}^l) \approx f(\vec{z}^l) + [\delta f/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ is a scalar, the product $[\delta f/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ must be coercible to a scalar, and therefore must have dimension 1×1. Recall that vectors $\vec{z}^l$, and their increments $\Delta\vec{z}^l$, are column vectors (i.e., they are matrices of dimension $N_l \times 1$).

Therefore, if $[\delta f/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ is defined using the standard inner product, gradient vectors $\delta f/\delta\vec{z}^l$ (for scalar functions f) have to be row vectors (with dimensions $1 \times N_l$); i.e, they must have dimension equal to those of the transpose of $\vec{z}^l$. This is an important point: derivatives of scalar functions with respect to column vectors are row vectors.

3.3 Derivative of a vector-valued function of a vector

Similarly, if $\vec{f} : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$is a vector-valued function, then $\vec{f}(\vec{z}^l + \Delta\vec{z}^l) \approx \vec{f}(\vec{z}^l) + [\delta\vec{f}/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ is a column vector with dimensions $N_{l+1} \times 1$. Therefore the product $[\delta\vec{f}/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ must be a column vector with $N_{l+1} \times 1$. It follows that if $[\delta\vec{f}/\delta\vec{z}^l] \cdot \Delta\vec{z}^l$ is defined using the standard matrix product, the matrix $[\delta\vec{f}/\delta\vec{z}^l]$ must have dimensions $N_{l+1} \times N_l$.

The matrix $[\delta\vec{f}/\delta\vec{z}^l]$, called the Jacobian matrix of $\vec{f}$, is the matrix whose i, j entry is the partial derivative $\delta f_i/\delta z_j$. We can also think of $[\delta\vec{f}/\delta\vec{z}^l]$ as being the set of (row) gradient vectors $\delta f_i/\delta\vec{z}^l$ (where f_i is the i-th component of the vector-valued function $\vec{f}$) stacked into a matrix with N_{l+1}rows.

3.4 Derivative of a scalar-valued function of a matrix

Next, suppose that $f : \mathbb{R}^{N_{l+1} \times N_l} \to \mathbb{R}$ is a scalar-valued function of the weight matrix W^{l+1} of dimension $N_{l+1} \times N_l$, whose elements are w_{ij}^{l+1}. Dropping the superscript $l + 1$ for the moment, we want to define $[\frac{\delta f}{\delta W}]$ in such a way that

$$f(W + \Delta W) \approx f(W) + [\frac{\delta f}{\delta W}] \cdot \Delta W.$$

But here we run into a problem. The expression $f(W + \Delta W)$ is a scalar, and therefore the expression $[\frac{\delta f}{\delta W}] \cdot \Delta W$ must also be a scalar. The matrix ΔW is, like W itself, an $N_{l+1} \times N_l$-dimensional matrix – but there is no matrix of any dimension which multiplies an $N_{l+1} \times N_l$ -dimensional matrix, using the standard matrix product, to create a 1×1 matrix that is coercible to a scalar.

The resolution to this problem is that the product needed in the expression $[\frac{\delta f}{\delta W}] \cdot \Delta W$ is not the standard matrix product. If W and ΔW are $N_{l+1} \times N_l$ -dimensional matrices, define $[\frac{\delta f}{\delta W}]$ to be the $N_l \times N_{l+1}$-dimensional matrix

$$[\frac{\delta f}{\delta W}]_{ji} = \frac{\delta f}{\delta w_{ij}},$$

for $j \in 1, ..., N_l$ and $i \in 1, ..., N_{l+1}$, and define

$$[\frac{\delta f}{\delta W}] \cdot \Delta W = \Sigma_{i=1}^{N_{l+1}} \Sigma_{j=1}^{N_l} \frac{\delta f}{\delta w_{ij}} \Delta w_{ij}. \tag{3}$$

This is actually a tensor product and contraction over both the row and column dimensions of ΔW (see Chapter 1.9 in [2]).

The thing to remember about this example is that if $f : \mathbb{R}^{N_{l+1} \times N_l} \to \mathbb{R}$ is a scalar-valued function of a matrix $W^l \in \mathbb{R}^{N_{l+1} \times N_l}$, then $[\frac{\delta f}{\delta W}]$ should be thought of as an $N_l \times N_{l+1}$-dimensional matrix; i.e., having the same shape as the transpose of W^l.

3.5 Derivative of a vector-valued function of a matrix

Finally, suppose that $f : \mathbb{R}^{N_{l+1} \times N_l} \to \mathbb{R}^M$ is a vector-valued function of the weight matrix W^{l+1} of dimension $N_{l+1} \times N_l$. Dropping the superscript $l + 1$ for W as before, we want to define $[\delta f / \delta W]$ in such a way that

$$\vec{f}(W + \Delta W) \approx \vec{f}(W) + [\frac{\delta \vec{f}}{\delta W}] \cdot \Delta W.$$

The dimensions of $\vec{f}(W)$ and ΔW are $M \times 1$ and $N_{l+1} \times N_l$, and there is no matrix that multiplies an $N_{l+1} \times N_l$-dimensional matrix to produce an $M \times 1$-dimensional vector. The product needed for $[\frac{\delta \vec{f}}{\delta W}] \cdot \Delta W$ is not a standard matrix product, and the object $[\frac{\delta \vec{f}}{\delta W}]$ is not a 2-dimensional matrix.

To get a feeling for what $[\frac{\delta \vec{f}}{\delta W}]$ needs to be, consider that if we confine ourselves to a single component f_k of the vector valued function $\vec{f}$, then $f_k : \mathbb{R}^{N_{l+1} \times N_l} \to \mathbb{R}$ is a scalar-valued function of the weight matrix W, and we are back in the case discussed in the previous section. Therefore, we should think of $[\delta f_k / \delta W]$ as being a matrix having the shape of the transpose of W, with

$$[\frac{\delta f_k}{\delta W}]_{ji} = \frac{\delta f_k}{\delta w_{ij}}.$$

Since $\vec{f}$ is a vector valued function with M components, for each $k \in 1...M$ there is a matrix $[\delta f_k / \delta W] \in \mathbb{R}^{N_l \times N_{l+1}}$. Since $\vec{f}(W)$ is itself an $M \times 1$-vector, we should think of $[\delta \vec{f}/\delta W]$ as being an $M \times 1$-dimensional column vector, each of whose components is the $N_l \times N_{l+1}$-dimensional matrix $[\delta f_k/\delta W]$.

To summarize, $[\delta \vec{f}/\delta W]$ is a '3-dimensional matrix', a 3-tensor, with dimensions $M \times N_l \times N_{l+1}$. Each element of $[\delta \vec{f}/\delta W]$ is indexed using 3 indices, $[k, i, j]$. The 2-dimensional 'slice' $[\delta \vec{f}/\delta W][k, \cdot, \cdot]$ through $[\delta \vec{f}/\delta W]$ is equal to the matrix $[\delta f_k/\delta W]$, which has dimensions $N_l \times N_{l+1}$. The product $[\frac{\delta \vec{f}}{\delta W}] \cdot \Delta W$ is a vector of dimension $M \times 1$ whose k-th component is $[\delta f_k/\delta W] \cdot \Delta W$, as defined in equation 3.

4 Derivation of backpropagation

To appreciate the backpropagation algorithm for computing the gradient of the cost function of a neural net – and by extension, the elegance of the neural net's construction – consider that a neural net may have millions of parameters, since every w^l_{ij} and every b^l_i for every layer l usually constitutes a separate parameter.

As calculus students, we've calculated gradients by hand of the form $\delta f/\delta \vec{p}$ for vectors $\vec{p}$ of dimensions up to, say, 5 or 10. You'll remember that every component of a gradient vector is a different function of all the parameters $\vec{p}$. If the neural net function in Equation 2 were not of a special form, calculating the derivative $\delta f/\delta p$ for every parameter p would be a grueling task. The cool thing about the deep neural network construction is that it is easy to calculate the gradients of its cost functions, but by varying the parameters W^l and b^l for all layers l, arbitrarily complicated functions can be expressed by a neural network (see chapter 4 in [1] for an explanation of this point).

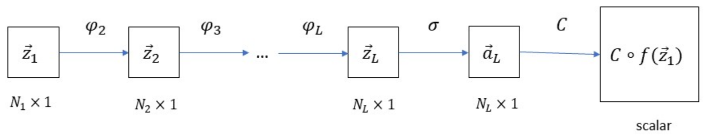

Figure 3: Another way to look at the neural net function definition.

Figure 3 shows another view of the function definition in equation 2. The input to the neural net is the training vector $\vec{z}^1$. It is first transformed sequentially by the functions $\phi_2, ..., \phi_L$ to produce the vector $\vec{z^L}$, which is then passed through the (vectorized) activation function σ to produce the final activation vector $f(\vec{z}^1) = \vec{a}^L$. The activation vector is then passed to the cost function C, which compares $f(\vec{z}^1)$ with its training label λ to calculate the cost function $C(f(\vec{z}^1), \lambda)$.

For each $l \in 1, ..., L$, define the scalar function $C_l : \mathbb{R}^{N_L} \to \mathbb{R}$ as

$$C_l = C \circ \sigma \circ \phi_L \circ \phi_{L-1} \circ ... \circ \phi_{l+1}. \tag{4}$$

It follows that $C_l = C_{l+1} \circ \phi_{l+1}$, and $C_1(\vec{z_i}^1) = C(f(\vec{z_i}^1), \lambda_i)$, where f is the neural network function defined in equation 2, and λ_i is the true value of $g(\vec{z}_i)$.

The objective of backpropagation is to obtain the partial derivatives $\delta C_1/\delta w^l_{ij}$ and $\delta C_1/\delta b^l_i$ for each layer l and all indices i, j.

Our approach will use the chain rule and the multidimensional derivative objects discussed in Section 3. This will be our strategy:

1. For layers $l = 1, ..., L$, define 'helper' gradients $\vec{\delta^l} \in \mathbb{R}^{N_l}$ by

$$\vec{\delta^l} = \delta C_l/\delta \vec{z^l},$$

   where $\vec{z^l}$ are the inputs to the layer l function ϕ_{l+1}, and C_l is as defined in equation 4. Since the $\vec{\delta^l}$ are gradients, we will think of them as row vectors with dimensions $N_l \times 1$ (see Section 2.2). These are easily calculated recursively for decreasing $l = L, ..., 1$.

   As an aside, the helper gradient $\vec{\delta^l}$ is referred to in [1] as the 'error in the layer l', because it is the gradient of the cost function with respect to the input $\vec{z^l}$ to the function ϕ_{l+1} in the l-th layer of the neural network.

2. Calculate the derivatives of the scalar function $C_1 = C \circ f$ with respect to weight matrices $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$ using the chain rule and the precomputed 'helper' gradients $\vec{\delta^l} \in \mathbb{R}^{N_l}$.

3. Calculate the derivatives of the scalar function C_1 with respect to the bias vectors $b^l \in \mathbb{R}^{N_l}$ using the chain rule and the precomputed 'helper' gradients $\vec{\delta^l} \in \mathbb{R}^{N_l}$.

4.1 Define the helper gradient $\vec{\delta^L}$ in the last layer of the neural net.

First, we must define the helper gradient $\vec{\delta^L} = \delta f/\delta \vec{z}^L$ for the last layer L in the neural net. The other helper gradients $\vec{\delta^l}$ will be calculated recursively from $\vec{\delta}^{l+1}$, working backward down the neural network to its beginning.

For the last layer L in the neural net, we have

$$\vec{\delta^L} = \delta(C \circ \sigma(\vec{z}^L))/\delta \vec{z}^L = \delta(C(\vec{a}^L))/\delta \vec{z}^L.$$

Applying the chain rule, we have

$$\vec{\delta^L} = [\frac{\delta C}{\delta \vec{a}^L}] \cdot [\frac{\delta \vec{a}^L}{\delta \vec{z}^L}],$$

where $[\delta C/\delta \vec{a}^L]$ is a row vector of dimension $1 \times N_L$, and $[\delta \vec{a}^L/\delta \vec{z}^L]$ is an $N_L \times N_L$-dimensional matrix. For most cost functions C, the gradient $[\delta C/\delta \vec{a}^L]$ with respect to the final activation vector $\vec{a^l}$ is simple to calculate. For example, if the cost function is the squared difference $C(\vec{a}^L) = \left\|\vec{a}^L - \vec{\lambda}\right\|^2$ between the neural network's output value $\vec{a}^L$and the true value $\vec{\lambda}$, then

$$\frac{\delta C}{\delta \vec{a}^L} = 2 \cdot (\vec{a}^L - \vec{\lambda})^t.$$

The derivative $[\delta \vec{a}^L / \delta \vec{z}^L]$, as derived in Section 2.3, is an $N_L \times N_L$-dimensional matrix, but it has a simple form; because a_j^L is a function of z_j^L only, the derivative is actually a diagonal matrix:

$$[\frac{\delta \vec{a}^L}{\delta \vec{z}^L}] = \text{diag}\{\sigma'(z_j^L)\}_{j=1}^{N_L}.$$

It follows that $\vec{\delta}_j^L = (\delta C / \delta a_j^L) \cdot \sigma'(z_j^L)$. Written in matrix product form,

$$\vec{\delta}^L = [\frac{\delta C}{\delta \vec{a}^L}] \cdot [\frac{\delta \vec{a}^L}{\delta \vec{z}^L}] = \frac{\delta C}{\delta \vec{a}^L} \cdot [\text{diag}\{\sigma'(z_j^L)\}_{j=1}^{N_L}]. \tag{5}$$

In [1], the same expression is written in vector form using the Hadamard (elementwise) vector product of $\delta C / \delta \vec{a}^L$ and $\sigma'(\vec{z}_L)$.

4.2 Define the helper gradient $\vec{\delta}^l$ as a function of $\vec{\delta}^{l+1}$.

Recall the scalar-valued functions $C_l : \mathbb{R}^{N_L} \to \mathbb{R}$ defined in equation 4. Our goal in this section is to calculate $\delta C_l / \delta \vec{z}^l$, given $\delta C_{l+1} / \delta \vec{z}^{l+1}$.

Since $C_l = C_{l+1} \circ \phi_{l+1}$, we have (according to the chain rule),

$$\vec{\delta}^l = [\frac{\delta C_l}{\delta \vec{z}^l}] = [\frac{\delta C_{l+1} \circ \phi_{l+1}}{\delta \vec{z}^l}] = [\frac{\delta C_{l+1} \circ \phi_{l+1}(\vec{z}^l)}{\delta \phi_{l+1}(\vec{z}^l)}] \cdot [\frac{\delta \phi_{l+1}(\vec{z}^l)}{\delta \vec{z}^l}] \tag{6}$$

$$= [\frac{\delta C_{l+1}}{\delta \vec{z}^{l+1}}] \cdot [\frac{\delta \vec{z}^{l+1}}{\delta \vec{z}^l}] = \vec{\delta}^{l+1} \cdot [\frac{\delta \vec{z}^{l+1}}{\delta \vec{z}^l}].$$

Let's review the dimensions and types of these objects for a moment. The helper gradient $\vec{\delta}^l$ is the gradient of C_l, a scalar function, with respect to $\vec{z}^l$, a vector; therefore it is a row vector of dimension $1 \times N_l$. Similarly, $\vec{\delta}^{l+1}$ is a row vector of dimension $1 \times N_{l+1}$.

The expression $[\delta \vec{z}^{l+1} / \delta \vec{z}^l]$ is the Jacobian of an N_{l+1}-vector valued function with respect to a N_l-vector valued argument, and therefore is a matrix with dimension $N_{l+1} \times N_l$. Multiplying a $1 \times N_{l+1}$ gradient vector on the right by a $N_{l+1} \times N_l$-dimensional matrix to get a $1 \times N_l$-dimensional gradient vector makes sense.

Now, it remains to calculate $[\delta \vec{z}^{l+1} / \delta \vec{z}^l] = [\delta \phi_l(\vec{z}^l) / \delta \vec{z}^l]$. Since $\phi_l(\vec{z}^l) = W^{l+1} \cdot \sigma(\vec{z}^l) + \vec{b}^{l+1}$, and W^{l+1} and $\vec{b}^{l+1}$ are constants with respect to $\vec{z}^l$, we have

$$[\delta \vec{z}^{l+1} / \delta \vec{z}^l] = [\delta(W^{l+1} \cdot \vec{a}^l + \vec{b}^l) / \delta \vec{z}^l] = W^{l+1} \cdot [\delta \vec{a}^l / \delta \vec{z}^l], \tag{7}$$

where $[\delta \vec{z}^{l+1} / \delta \vec{z}^l]$ is a matrix of dimension $N_{l+1} \times N_l$, W^{l+1} is a matrix of dimension $N_{l+1} \times N_l$, and $[\delta \vec{a}^l / \delta \vec{z}^l]$ is a matrix of dimension $N_l \times N_l$.

Since $(\vec{a}_l)_j$ is a function of $(\vec{z}_l)_j$ only, $[\delta\vec{a}^l/\delta\vec{z}^l]$ is actually a diagonal matrix for all l:

$$[\delta\vec{a}^l/\delta\vec{z}^l] = \text{diag}\{\sigma'(z_j^l)\}_{j=1}^{N_l}. \tag{8}$$

Putting together equations 6,7, and 8, we get:

$$\vec{\delta}^l = \vec{\delta}^{l+1} \cdot W^{l+1} \cdot \text{diag}\{\sigma'(z_j^l)\}_{j=1}^{N_l}, \tag{9}$$

where the products are standard matrix-vector products. This formula differs slightly from the equivalent formula (BP2) in [1] because in this treatment, the vectors $\vec{\delta}^l$ are gradients and therefore row vectors.

4.3 Calculate the derivatives of the scalar function $C_1 = C \circ f$ with respect to the weight matrices $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$.

Now we have come to the core goal of backpropagation: we need to be able to find the gradient of $C_1 = C \circ f$ with respect to all of the weight matrices $\{W^l\}$ and bias vectors $\{\vec{b}^l\}$.

Again, according to the chain rule (with an appropriately defined product), we write

$$[\frac{\delta C_1}{\delta W^l}] = [\frac{\delta C_1}{\delta \vec{z}^l}] \cdot [\frac{\delta \vec{z}^l}{\delta W^l}]. \tag{10}$$

The first term, $[\delta C_1/\delta\vec{z}^l]$, is a derivative of a scalar function with respect to a vector argument, so it is a gradient and therefore a row vector. In fact, since

$$[\frac{\delta C_1}{\delta \vec{z}^l}] = [\frac{\delta(C_l \circ \phi_{l-1} \circ \phi_{l-2} \circ ... \circ \phi_1)}{\delta(\phi_{l-1} \circ \phi_{l-2} \circ ... \circ \phi_1(\vec{z}^1))}] = [\frac{\delta C_l}{\delta \vec{z}^l}],$$

it follows that

$$[\delta C_1/\delta\vec{z}^l] = [\delta C_l/\delta\vec{z}^l] = \vec{\delta}^l, \tag{11}$$

where $\vec{\delta}^l$ is the helper gradient of Section 3.3.

The term $[\delta\vec{z}^l/\delta W^l]$ is a derivative of a vector-valued function with respect to a matrix-valued valued argument, defined by:

$$[\frac{\delta \vec{z}^l}{\delta W^l}] = \frac{\delta}{\delta W^l}(W^l \cdot \sigma(\vec{z}^{l-1}) + \vec{b}^l) = \frac{\delta}{\delta W^l}(W^l \cdot \vec{a}^{l-1} + \vec{b}^l).$$

Referring back to Section 2.5, $[\delta\vec{z}^l/\delta W^l]$ is actually a '3-dimensional matrix' (i.e., a 3-tensor) with dimensions $N_l \times (N_{l-1} \times N_l)$. The 2-dimensional 'slice' $[\delta\vec{z}^l/\delta W^l][k,\cdot,\cdot]$ is equal to the matrix $[\delta z_k^l/\delta W]$, where z_k^l is the k-th coordinate of $\vec{z}^l$.

If $w_{i,j}^l$ is the (i,j)- coordinate of the weight matrix W^l, then we have the following expression for the (k,i,j) term of $[\delta\vec{z}^l/\delta W^l]$:

$$[\frac{\delta \vec{z}^l}{\delta W^l}]_{(k,i,j)} = \frac{\delta z_k^l}{\delta w_{j,i}^l} = \begin{cases} \frac{\delta}{\delta w_{ji}^l}(\Sigma_i w_{ji}^l a_i^{l-1} + b_j^l) = a_i^{l-1} & \text{if } k = j \\ 0 & \text{otherwise.} \end{cases}$$

It follows that

$$[\frac{\delta \vec{z}^l}{\delta W^l}]_{(k,i,j)} = I_{j=k} \cdot a_i^{l-1},$$

where $I_{j=k}$ indicates the Kronecker delta (i.e, it is 1 if $j = k$, and otherwise 0).

We calculate the final product from equation 10, which is given by a tensor product of the gradient vector $[\delta C_1/\delta \vec{z}^l]$ with the 3-tensor $[\delta \vec{z}^l/\delta W^l]$:

$$[\frac{\delta C_1}{\delta W^l}]_{(i,j)} = \Sigma_{k=1}^{N_l}[\delta C_1/\delta \vec{z}^l]_k \cdot [\delta \vec{z}^l/\delta W^l]_{(k,i,j)} \tag{12}$$

$$= \Sigma_{k=1}^{N_l} \delta_k^l \cdot I_{j=k} \cdot a_i^{l-1} = \delta_j^l \cdot a_i^{l-1}.$$

Thus, the derivative of the neural network cost function with respect to the weight matrix W^l is the matrix that is the (outer) tensor product of the helper gradient $\vec{\delta^l}$ with the activations $\vec{a}^{l-1}$ from the previous layer. This product is written in tensor product notation as

$$[\frac{\delta C_1}{\delta W^l}] = \vec{a}^{l-1} \otimes \vec{\delta^l}. \tag{13}$$

4.4 Calculate the derivatives of the scalar function $C_1 = C \circ f$ with respect to the bias vectors $b^l \in \mathbb{R}^{N_l}$.

By the chain rule, we have

$$[\frac{\delta C_1}{\delta \vec{b^l}}] = [\frac{\delta C_1}{\delta \vec{z}^l}] \cdot [\frac{\delta \vec{z}^l}{\delta \vec{b^l}}]. \tag{14}$$

As shown in the previous section, $[\delta C_1/\delta \vec{z}^l] = \vec{\delta^l}$, the helper gradient in layer l.

$[\delta \vec{z}^l/\delta \vec{b^l}]$ is a derivative of an N_l-vector valued function by an N_L-vector valued argument, so it can be thought of as an $N_l \times N_l$-dimensional matrix. Since

$$\frac{\delta}{\delta \vec{b^l}}(W^l \cdot \sigma(\vec{z}^{l-1}) + \vec{b^l}) = \frac{\delta \vec{b^l}}{\delta \vec{b^l}} = I$$

is the $N_l \times N_l$ identity matrix, we have

$$[\frac{\delta C_1}{\delta \vec{b^l}}] = [\frac{\delta C_1}{\delta \vec{z}^l}] \cdot [\frac{\delta \vec{z}^l}{\delta \vec{b^l}}] = \vec{\delta^l} \cdot I = \vec{\delta^l}. \tag{15}$$

So the gradient of C_1 with respect to the bias vector $\vec{b^l}$ is exactly the helper gradient $\vec{\delta^l}$.

5 Summary

Once we are able to calculate $[\delta C_1/\delta W^l]$ and $[\delta C_1/\delta \vec{b}^l]$ for each l, we have the entirety of the gradient of C_1 with respect to all the parameters in the neural network. It is easy to write out the pseudocode for a feed-forward and backpropagation loop with a single training example $(\vec{z}^1, \lambda)$:

1. Input the training sample $\vec{z}^1$ to the neural network.
2. Feed-forward: For $l = 1, ..., L$, iteratively calculate (and save) $\vec{a}^l = \sigma(\vec{z}^l)$ and $\vec{z}^{l+1} = W^{l+1} \cdot \vec{a}^l + \vec{b}^{l+1}$.
3. Backpropagation step 1: Compute the L-th helper gradient $\vec{\delta}^L$:

$$\vec{\delta}^L = [\frac{\delta C}{\delta \vec{a}^L}] \cdot [\frac{\delta \vec{a}^L}{\delta \vec{z}^L}] = \frac{\delta C}{\delta \vec{a}^L} \cdot [\text{diag}\{\sigma'(z_j^L)\}_{j=1}^{N_L}].$$

4. Backpropagation step 2: For each $l = L-1, ..., 1$, compute $\vec{\delta}^l$ recursively from $\vec{\delta}^{l+1}$:

$$\vec{\delta}^l = [\frac{\delta C_{l+1}}{\delta \vec{z}^{l+1}}] \cdot [\frac{\delta \vec{z}^{l+1}}{\delta \vec{z}^l}] = \vec{\delta}^{l+1} \cdot W^{l+1} \cdot \text{diag}\{\sigma'(z_j^l)\}_{j=1}^{N_l}.$$

5. Backpropagation step 3: For each $l = L, ..., 2$, compute $[\frac{\delta C_1}{\delta W^l}]$ as the outer tensor product of the activation (column) vector $\vec{a}^{l-1}$ with the gradient (row) vector $\vec{\delta}^l$:

$$[\frac{\delta C_1}{\delta W^l}] = [\frac{\delta C_1}{\delta \vec{z}^l}] \cdot [\frac{\delta \vec{z}^l}{\delta W^l}] = \vec{a}^{l-1} \otimes \vec{\delta}^l.$$

.

6. Backpropagation step: For each $l = L, ..., 2$, compute

$$[\frac{\delta C_1}{\delta \vec{b}^l}] = [\frac{\delta C_1}{\delta \vec{z}^l}] \cdot [\frac{\delta \vec{z}^l}{\delta \vec{b}^l}] = \vec{\delta}^l.$$

References

[1] Neilsen, Michael, *Neural Networks and Deep Learning.* Free online book: downloaded at *http://neuralnetworksanddeeplearning.com/.*

[2] Porat, Boaz, *A Gentle Introduction to Tensors.* Downloaded at: *https://www.ese.wustl.edu/~nehorai/Porat_A_Gentle_Introduction_to_Tensors_2014.pdf*

[3] 3Blue1Brown, *Backpropagation calculus | Deep learning, chapter 4.* Video viewable at: https://www.youtube.com/watch?v=tIeHLnjs5U8

[4] Petersen, K.B., Pedersen, M.S., *The Matrix Cookbook.* Downloaded at: *https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf.*

[5] Minka, T.P., *Old and New Matrix Algebra Useful for Statistics.* Downloaded at: *https://tminka.github.io/papers/matrix/minka-matrix.pdf*