Machine Learning Guide for Petroleum Professionals: Part 3

Image by Shutterstock

Welcome back! I am excited you are here. We have covered a lot of ground in part 1 and part 2 of this series. Those were appetizers, where we explored some of the basics you'd need to know on our journey to understand how to deploy machine learning in the petroleum industry. This part 3 is our hearty main course, where we will delve into concepts of deep learning and explore the mathematics behind it. It's a very good idea to eat your appetizer before the main meal, so do head over to those earlier articles, if you haven't read them yet, or refresh your memory, before continuing here. With that said, let's jump into the Part 3 where we will explore the world of deep learning.

In part one, we discussed the four steps of a machine learning algorithm. They are:

1) initialize parameters,
2) compute cost function,
3) compute gradient descent and update it, and
4) repeat steps 2 and 3.

In our linear regression model, we initialized parameters (w and b) with zero, which was fine for that case. However, for a neural network (NN), we cannot initialize w with zero but rather with a random value. If you are wondering why we cannot initialize w with zero, the answer is simply one word: math. If you want to dig deeper into this math, read this post. If you skip this math, it will not hurt your machine learning understanding. Just remember that it's necessary to start w with a small random number (not zero and not too large) while b can be initialized with zero. The question then becomes how to initialize w with a random value.

He et al. (2015) derived a robust initialization method for w which is: initialize w with a small random value and multiply it with sqrt(2/no. of neurons in the previous layer) or √(2/no. of neurons in the previous layer). This method ensures that the weights of the neurons in each layer of the neural network are neither too small nor too large, which helps prevent vanishing or exploding gradients during training. Vanishing gradients happen when early layer gradients (derivatives) are too small, leading to slow or no weight updates, while exploding gradients happen when early layer gradients are too large, causing unstable training.

Mathematically, the initialization method proposed by He et al. (2015) is:

$$w = Small\ Random\ Value * \sqrt{\frac{2}{number\ of\ neurons\ in\ previous\ layer}} \rightarrow \text{Eq. 1}$$

For the first hidden layer, the number of neurons in the previous layer is the number of features of input while for the other layers, it is self-explanatory. More about it will be discussed later. For now, let's revise the two hidden layers NN algorithm (configuration), which we explored in part 2, by visualizing Figure 1 below. Here, X is the input to the first hidden layer (with one neuron) and a1 is its output. The a1 then becomes the input to the second hidden layer which gives a2 as the output. The a2 is then fed to the final layer (output layer) which gives the ŷ.

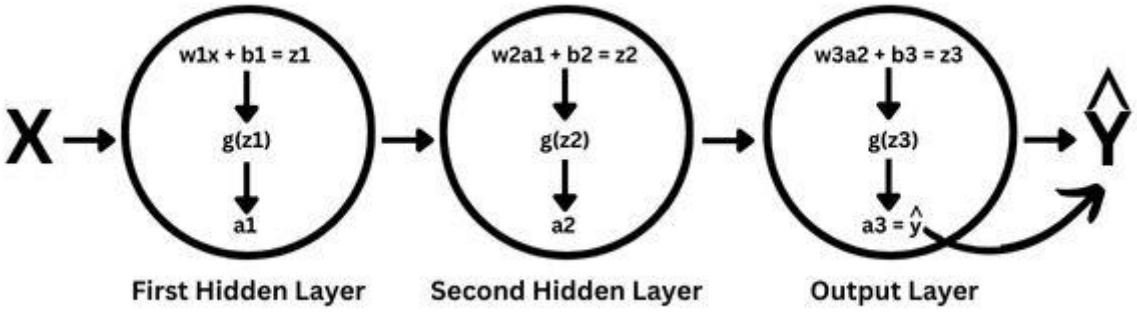

Figure 1: Two Hidden Layer Model. Image by Author

In Figure 1, the neural network architecture has a total of three layers: two hidden layers and one output layer. It's important to note that, in machine learning convention, the input layer is not counted as a layer. So, if a model has five layers in total, it means that there are four hidden layers and one output layer.

It's also important to note that in machine learning, w and b are typically represented as vectors (matrices) rather than scalars (single digits). Therefore, it's essential to understand how to perform basic operations such as multiplication and addition with matrices. To multiply two matrices, the number of columns in the first matrix must match the number of rows in the second matrix. Then, the product of the two matrices is obtained by taking the dot product of each row in the first matrix with each column in the second matrix. The result is a new matrix with the same number of rows as the first matrix and the same number of columns as the second matrix. For a visual representation of matrix multiplication, see Figure 2 below.

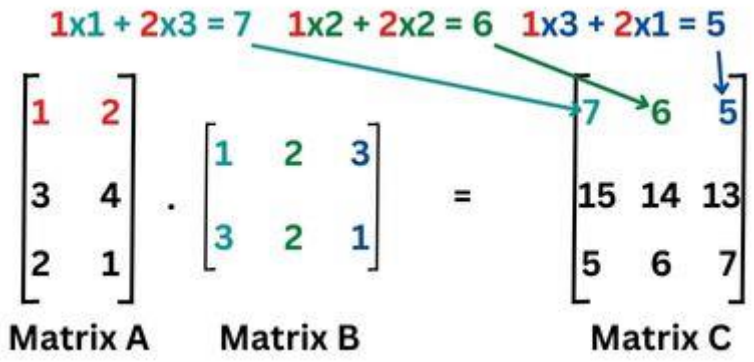

Figure 2: Matrix Multiplication. Image by Author.

Adding or subtracting matrices is a straightforward operation. Simply add or subtract the corresponding elements of each matrix. If you need to add a scalar value (a single digit) to a matrix, you can do so by adding that scalar value to each element of the matrix. Figure 3 below provides a visual representation of these operations.

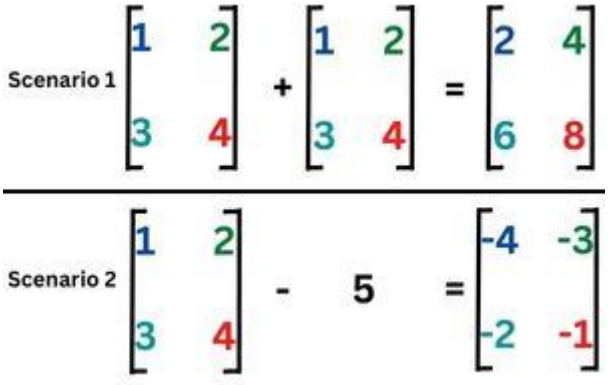

Figure 3: Matrix Add/Subs. Image by Author.

Now that you understand matrix operations such as multiplication and addition, as well as how to initialize the parameters in a neural network, it's important to clarify the notation used to represent matrices and vectors. In particular, we will use capital letters such as W to represent matrices, and lowercase letters such as w to denote their individual components. For example, W1 is the weight matrix for the first hidden layer, and it contains values w1, w2, w3, etc., which correspond to the first, second, and third neuron of that layer. Similarly, we use A1 to represent the output of the first hidden layer in matrix form, with a1, a2, a3, etc. representing the individual values inside that matrix. The same notation is used for A2, A3, and so on. However, in machine learning convention, matrix b is also represented by b (not B). Figure 4 below provides a visual representation of this concept for a neural network with two hidden layers, each containing two neurons, and an output layer with one neuron.

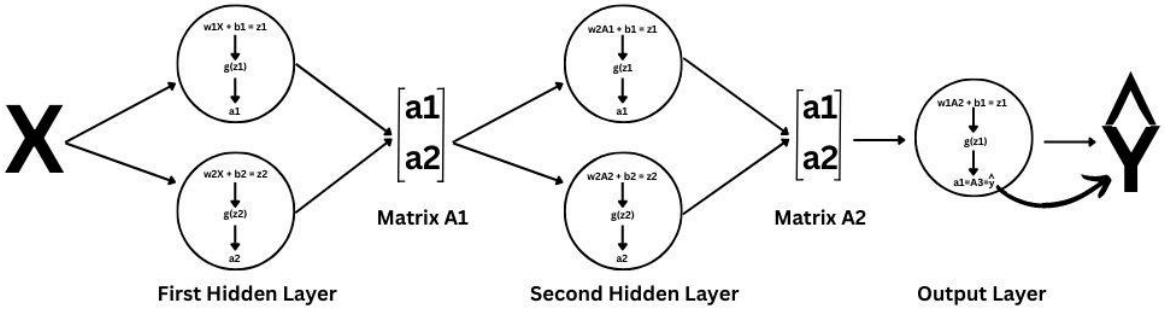

Figure 4: Hidden Layers with Matrix concept

Note that in the above figure, the values of w1, and w2 of the first hidden layer are different from the values of w1 and w2 in the second hidden layer. In machine learning, we have different symbols for both, but I decided to keep things simple by not introducing more notations. Same goes for z1, z2, a1, and a2.

Now it's to time discuss one of the paramount concepts: dimensions of X, W, and b. The dimension of X depends on the number of features (n) and the number of samples (m). Similarly, the dimension of W depends on the number of neurons and the number of features. The features are the parameters on which output is dependent. Let's say we have a hundred values of density log, neutron log, and sonic log, and a hundred values of their corresponding permeability. If these logs are the input to a machine learning model and the output is the permeability, then the number of features is three (density, neutron, and sonic) and denoted by n. And, as discussed in part 1, the number of examples, inputs, X, and samples (different names for the same thing) is 100, denoted by m.

In the case of the input matrix X, the number of rows should equal the number of features (n), and the number of columns should equal the number of samples (m). For example, if we have 100 values of porosity as input to a neural network, then the dimension of X would be 1 by 100, one feature (porosity) and 100 samples.

For the first hidden layer, the number of rows in W1 should equal the number of neurons in that layer, and the number of columns should equal the number of features in the input. For example, if the first hidden layer contains three neurons and we have one feature (porosity) in the input, then W1 should be a 3 by 1 matrix.

Similarly, for the second and subsequent hidden layers, the number of rows in the weight matrix (W) should be equal to the number of neurons in that layer, and the number of columns should be equal to the number of neurons in the previous layer. For example, for W3, the number of rows should be equal to the number of neurons in the third hidden layer, and the number of columns should be equal to the number of neurons in the second hidden layer.

For bias (b), the number of rows should equal the number of neurons in that layer, and the number of columns should always be 1.

Once we have initialized the correct dimensions for W and b, the dimensions for the output matrices Z and A will automatically be calculated.

To further concrete this concept, let me give you an example. Suppose we have input X as porosity and output Y as permeability as shown in Table 1 below. Our task is to train the NN model to predict permeability.

Porosity (X)	Permeability (Y)
13.76	193.72
9.60	105.71
11.66	138.53

Here, the number of features, n, is one (porosity) and the number of inputs or number of examples, m, is three. Let's create a three-layer neural network (two hidden layers and one output layer). Let's assume that each hidden layer has three neurons and the output layer has only one neuron as shown in Figure 5 below.

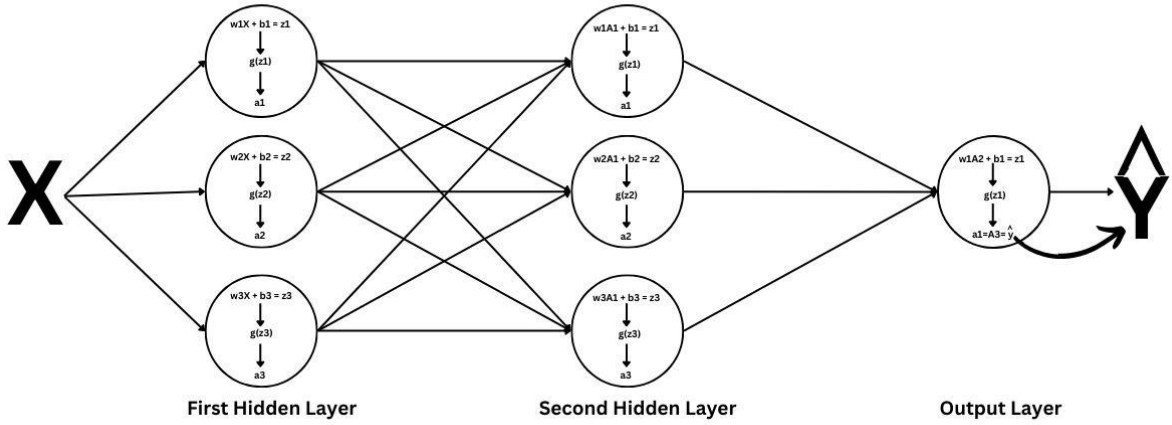

Figure 5: Three-layers Neural Network

So, the dimension of X should be 1 by 3 (the number of features, number of examples). W1 should be 3 by 1 (the number of neurons of the first hidden layer which is three in this case, and the number of features which is one, porosity, in this case. Similarly, the shape of b1 should be 3 by 1, as the number of rows should be equal to the number of neurons of the first hidden layer and the number of columns should be equal to one. By the same intuition, we can determine the dimension of the second and third layers. To familiarize yourself with

these concepts, see Figure 6.

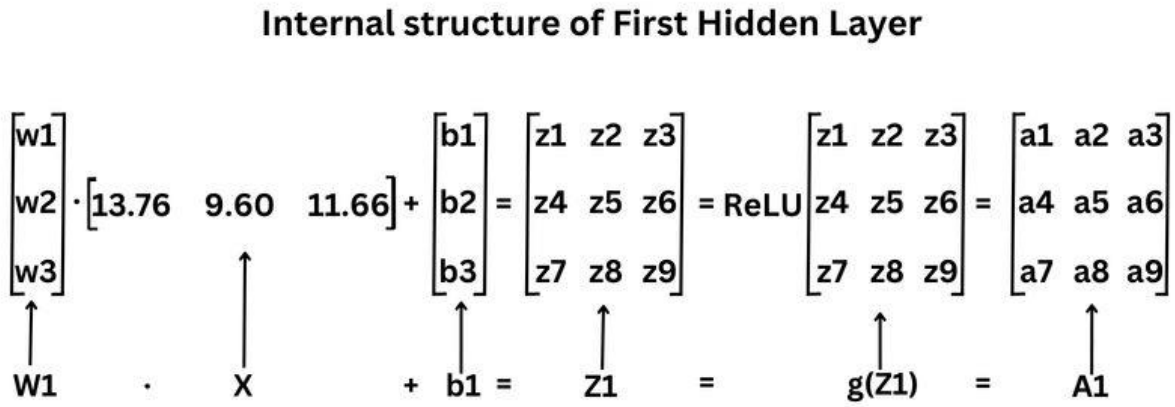

Figure 6: Internal structure of First Hidden Layer.

The above Figure 6 represents the internal structure of the first hidden layer. Firstly, we multiply matrix W1 with matrix X. Remember the general rule of matrix multiplication: the shape of a new matrix will have the same number of rows as the first matrix and the same number of columns as the second matrix. So, the resultant matrix after a dot product of W1 and X has the shape of 3 by 3 (not shown in the above figure). Then we add b1 to get Z1. The shape is still 3 by 3. Then we apply the ReLU activation function (discussed in part 2) to Z1, denoted by g(Z1), to get A1. The shape is still 3 by 3. That A1 is now input to the second hidden layer.

Up to this point, you have acquired an understanding of how to initialize parameters and the dimensions of all the matrices involved in the process. Now let's proceed with implementing the four steps of machine learning as discussed in part 1, step by step, using the dataset presented in Table 1.

Step 1: Initialize parameters

We will be using the same three-layer model as shown in Figure 5 and the data shown in Table 1. As a recap, the shape of our input X (porosity) is 1 by 3, while the shape of W1 is 3 by 1 and the shape of b1 is 3 by 1. Similarly, for the second layer, the shape of input A1 is 3 by 3, W2 is 3 by 3, and b2 is 3 by 1. The shape of the last layer, which produces the predicted permeability (A3 or $\hat{Y}$), is 1 by 3. It's crucial to understand this concept, so if you're not clear on it, please reread all the text after Figure 5.

Now let's initialize the values of W and b. As previously mentioned, we'll use the method proposed by He et al. (2015) to initialize the weight (see equation 1). For simplicity, I assume that "small random value" is equal to 1 (but for real problems, it is recommended to initialize the weights with different random values, not just 1). Therefore, initializing W will just be sqrt(2/no. of neurons in the previous layer). Since our input X has only one feature, for W1, no. of neurons in the previous layer is equal to 1. So all the values of W1 are initialized to √(2/1) = 1.41. For the second hidden layer, no. of neurons in the previous layer (first layer) is equal to 3, so all the values of W2 are initialized to √(2/3) = 0.81. Finally, for the output layer, no. of neurons in the previous layer (second layer) is equal to 3, so all the

values of W3 are initialized to √(2/3) = 0.81. We can start all the values of b with zero. Check out Figure 7 below for a visualization of all the values of W and b.

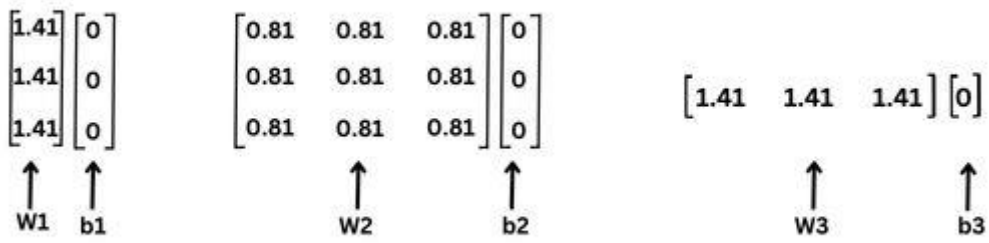

Figure 7: Initialize W and b

Step 2: Cost Function

From part 1, we are aware that to find the cost function, we need to first find Ŷ. And for Ŷ, we need to do the calculations of all the layers of our neural network. See Figure 8 below for the calculation of the first hidden layer.

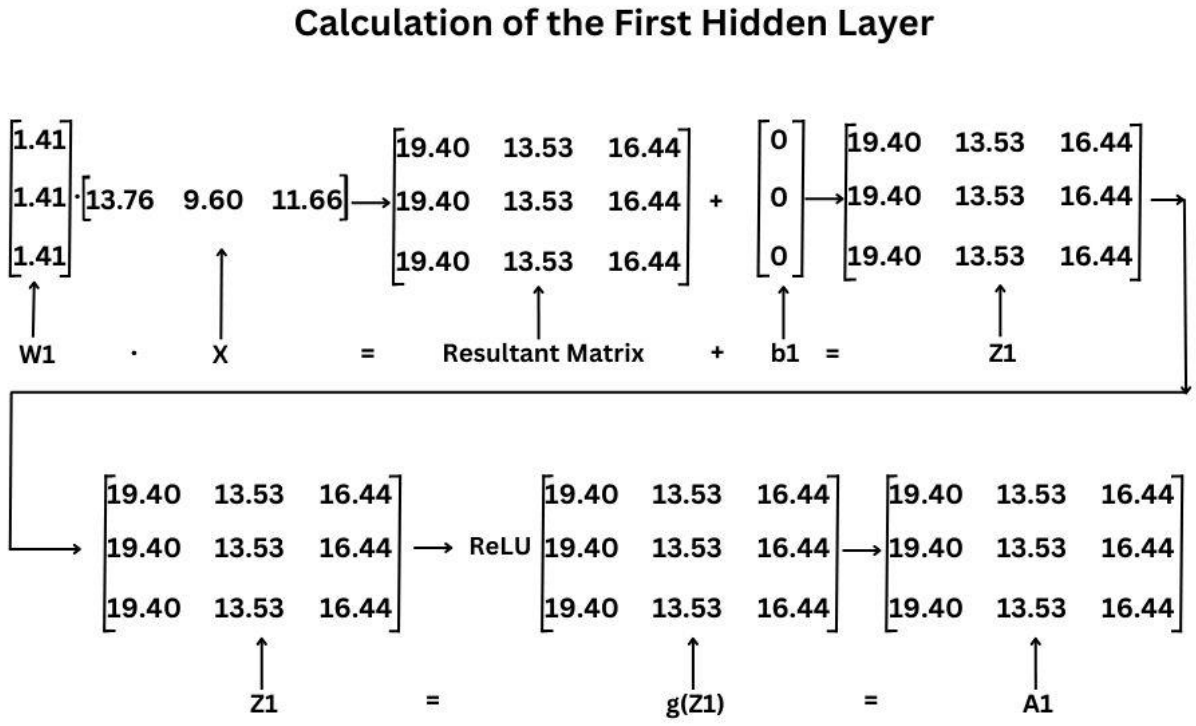

Figure 8: Calculation of First Hidden Layer

Now we have A1, the output of the first hidden layer. We will feed this into the second hidden layer as an input. Recall that the weight matrix W2 has dimensions 3 by 3, with values of 0.81, and the bias matrix b2 has dimensions 3 by 1, with all zeroes. Figure 9 below shows the calculation of the second hidden layer.

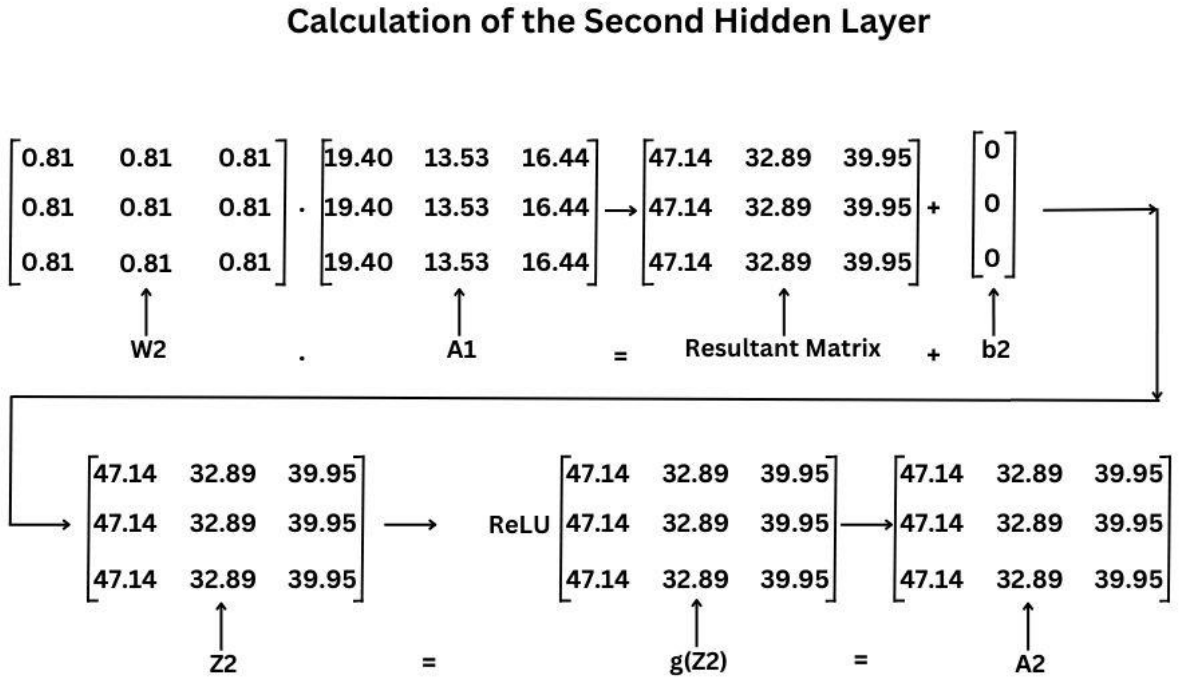

Figure 9: Calculation of the Second Hidden Layer

Now it's time to feed A2 to the last layer. We know that W3 is 1 by 3 matrix with values of 0.81 and b3 is 1 by 1 matrix with zero. See Figure 10 below for a visual representation of the last layer's calculation.

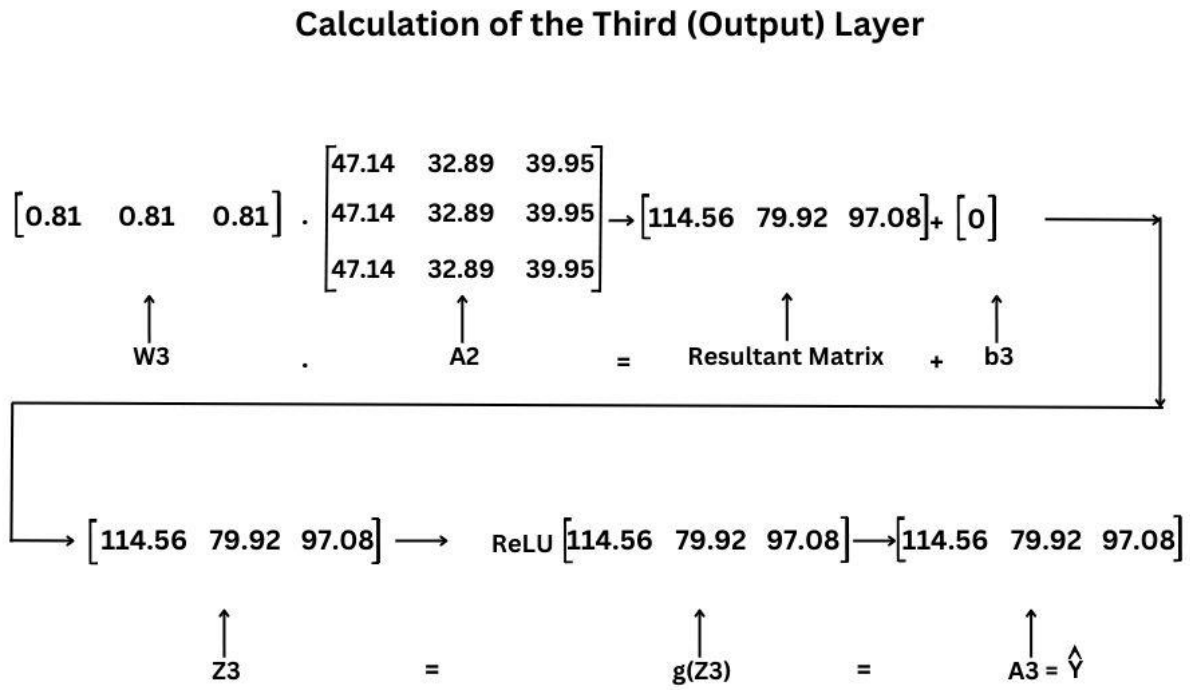

Figure 10: Calculation of the Final Layer

So, finally, we have our last output, $\hat{Y}$. This is our predicted permeability in a matrix form, at a given value of porosity. Now it's time to find the cost function. As we learned in part 1, the cost function J is:

$$J = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{Y} - Y\right)^2$$

We now have our predicted permeability values, denoted as $\hat{Y}$, and the actual permeability values, denoted as Y, both in vector form (matrices), so $\hat{y}$ and y are the values inside the $\hat{Y}$ and Y, respectively. We can find the difference between the two vectors by subtracting the values elementwise. In the case of this example, $\hat{Y}$ is [114.56, 79.92, 97.08] and Y, from Table 1, is [193.72, 105.71, 138.53], so the difference is [-79.15, -25.78, -41.44]. We can then square each element of this difference vector to obtain [6264.72, 664.60, 1717.27]. The cost function J, as defined in Part 1, is obtained by summing up these squared differences and dividing them by 2m, where m is the number of samples, which is 3 in this case. Thus, J = (12467.95 + 2348.37 + 4759.62) / (2 * 3) = 1441.09. This cost is quite high, indicating that our model is not yet accurately predicting permeability. Here comes the third step to decrease the cost.

Step 3: Gradient Descent

As discussed in part 1, we need to take small steps to decrease the cost. We will do that by finding the current value of slope (derivative) and then adjusting it slowly to find the updated values for W and b. In part 1, we determined the derivative of J with respect to

(w.r.t) w and b because we had only one layer. But here, we have three layers, so, we need to find the derivatives of A, Z, W, and b of all layers, starting from the last (output) layer and moving backward to the hidden layers (second and first layer).

From part 1, we are aware of Mean Square Error (MSE) which is also called the Loss function. Let's denote it by L. Here, A3 is the "result of the last layer" which is also equal to Ŷ. See the below equation.

$$L = \frac{\left(\hat{Y} - Y\right)^2}{2}$$

Now take the derivative of L w.r.t Ŷ. Let's denote the derivative term by dA3 (d for the derivative of L and A3 means w.r.t A3). So, the derivative of the above equation is:

$$dA3 = \left(\hat{Y} - Y\right)$$

Now it's time to find the derivative of L w.r.t Z3, denoted by dZ3. It is equal to dA3*g' where g' shows the derivative of ReLU and is read as g prime. One point to note is that g' is a vector (matrix). In the case of ReLU, all the values of g' are equal to 1 for all the positive values of Z, and equal to zero for all negative and zeroes. In this case, we have only positive values in Z3, so, all the values of g' are equal to 1. In other words, dZ3 is equal to dA3 since the derivative of ReLU with respect to Z3 is equal to 1 for all positive values. Therefore, the derivative of L w.r.t Z3 is:

$$dZ3 = dA3$$

And the derivatives of L w.r.t W3 and b3, denoted by dW3 and db3 respectively, are:

$$dW3 = \frac{dZ3.A2^T}{m}$$

$$db3 = \frac{\sum dZ3}{m}$$

Where T in dW3 means transpose of vector A2 and ∑ in db3 shows the sum of all the values of dZ3 (kept in matrix forms).

If you're interested in learning more about how we derive these derivatives, you can check out the linked video after completing this article. However, if you're not interested in the calculus behind it, you can skip the video and it will not affect your overall understanding of deep learning.

But it is recommended that you see all the derivative calculations of the third layer in Figure 11 below.

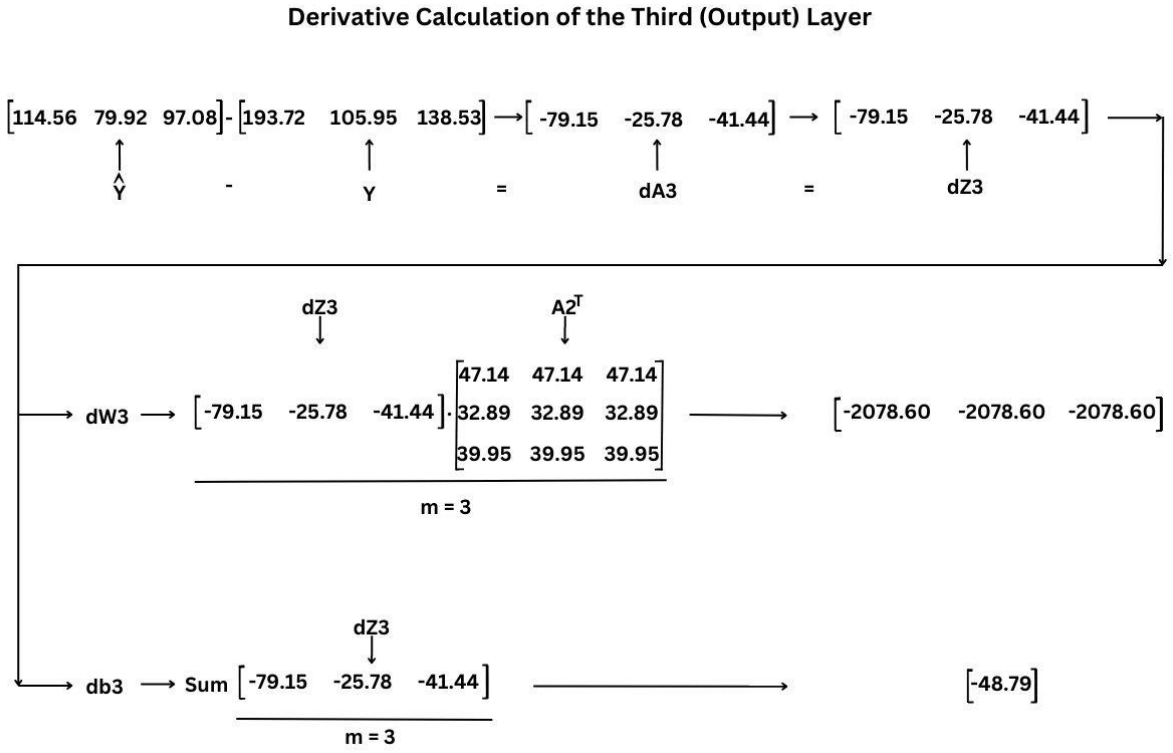

Figure 11: Derivative calculation of the Output Layer.

Now it's time to calculate the derivative for the hidden layers. The steps are similar to the output layer, except for the "dA" term. The following are the derivatives for the second hidden layer, and Figure 12 shows the calculations:

$$dA2 = W3^T . dZ3$$

$$dZ2 = dA2$$

$$dW2 = \frac{dZ2 . A1^T}{m}$$

$$db2 = \frac{\sum dZ2}{m}$$

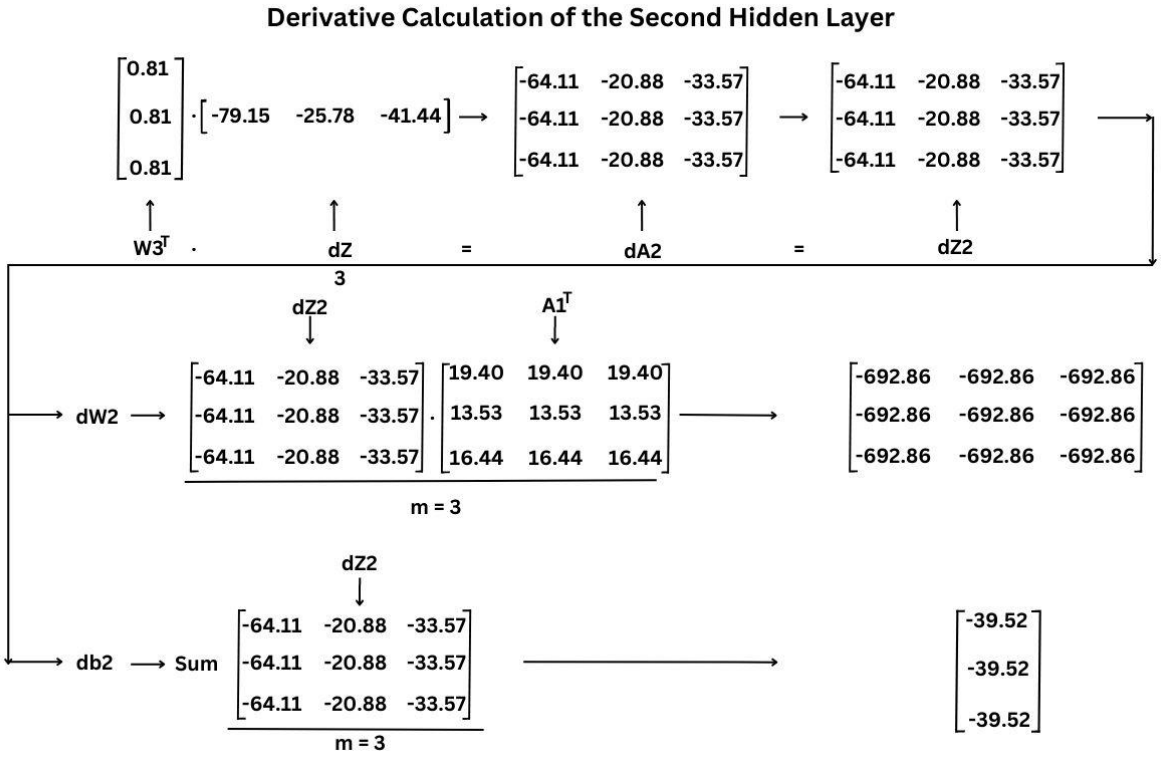

Figure 12: Derivative calculation of the Second Hidden Layer.

The derivatives for the first hidden layer are as below and Figure 13 shows calculations:

$$dA1 = W2^T . dZ2$$

$$dZ1 = dA1$$

$$dW1 = \frac{dZ1 . X^T}{m}$$

$$db1 = \frac{\sum dZ1}{m}$$

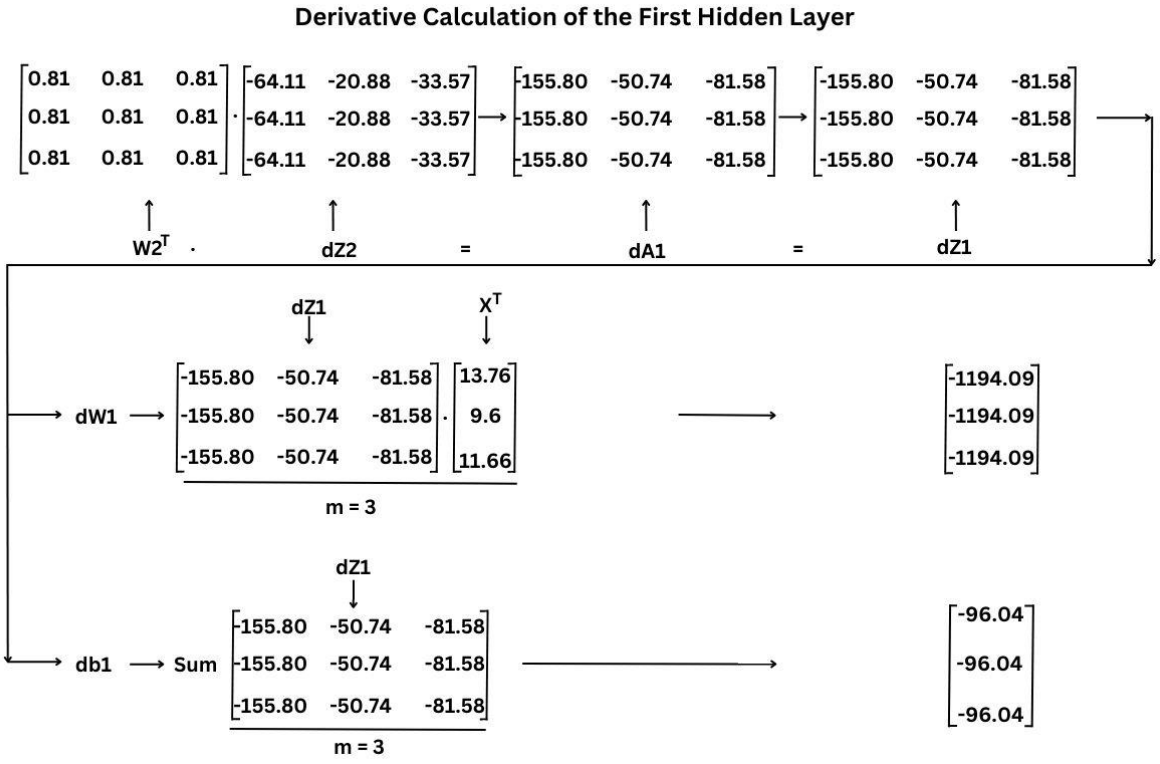

Figure 13: Derivative calculation of the First Hidden Layer.

Now that we have calculated the derivative values for all parameters (W and b), it's time to update them (update gradient descent). As a quick reminder, the formulae from part 1 are:

$$w_{(updated)} = w_{(current)} - \alpha * \frac{\partial \bar{J}}{\partial w}$$

$$b_{(updated)} = b_{(current)} - \alpha * \frac{\partial \bar{J}}{\partial b}$$

Let's rewrite them according to the above notations.

$$W3_{(updated)} = W3_{(current)} - \alpha * dW3$$

$$b3_{(updated)} = b3_{(current)} - \alpha * db3$$

$$W2_{(updated)} = W2_{(current)} - \alpha * dW2$$

$$b2_{(updated)} = b2_{(current)} - \alpha * db2$$

$$W1_{(updated)} = W1_{(current)} - \alpha * dW1$$

$$b1_{(updated)} = b1_{(current)} - \alpha * db1$$

Assume α is 0.01. See all the calculations in Figure 14 below.

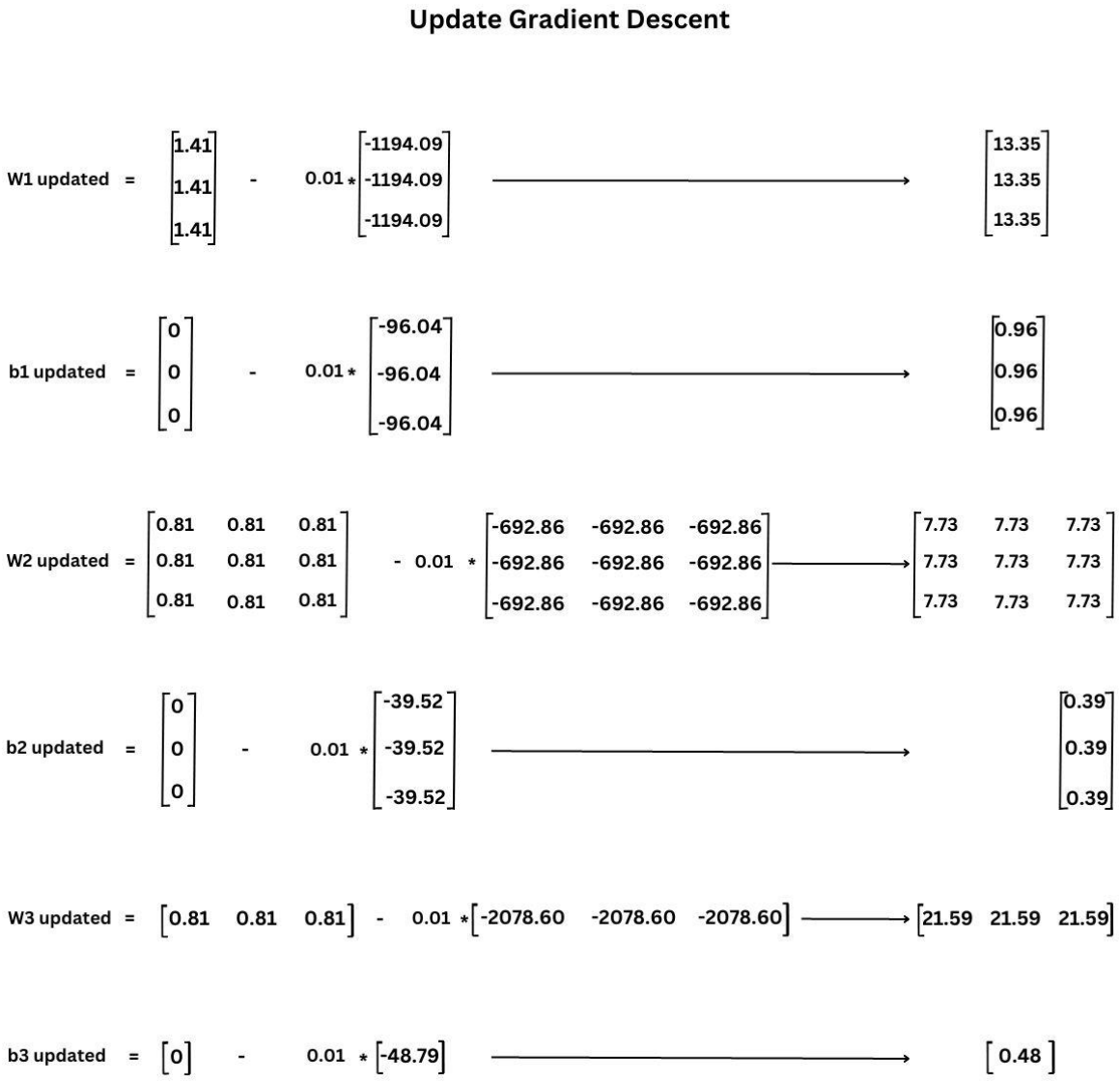

Figure 14: Update Gradient Descent

Congrats. You have successfully completed the first iteration of the three-layer deep neural network. You should be proud of yourself. Now that we have updated the values for the parameters, we can use these values to repeat the step 2 and 3 several times to decrease the cost function. This repetitive and monotonous task can be done in Python in a matter of seconds. We will do this in part 4 with a full set of real field data. We will also go through the data scaling and data distribution (training and testing set) and generalize our intuition to L-Layer neural network, where L is any number of layers in a model.

Thank you for reading all the way to the end, and I hope to see you in part 4!

Read part 4 here.