Machine Learning Guide for Petroleum Professionals: Part 4

Welcome to the final part of our machine learning journey. In Part 1, we covered some basics of machine learning, while in Part 2, we delved into non-linear activation functions. In Part 3, we explored the fascinating world of deep learning.

In this concluding part, we'll be taking our understanding to the next level by discussing the L-layers model, exploring some essential data preprocessing techniques, and understanding the training and testing data sets. We'll then tie everything together by applying our newfound knowledge to a real-world oil and gas case study using a complete set of data. By the end, you will be fully equipped to apply machine learning to real-world oil and gas problems and make accurate predictions. So, let's dive in without further ado!

In part 3, we explored the three layers model (two hidden layers and one output layer). We also discussed the notations, forward steps, and backward steps for each layer. Just a recap, the forward propagation of the second hidden layer is:

$$Z2 = W2.A1 + b2$$

$$A2 = g(Z2)$$

And backward propagation (derivatives) for the same (second) layer is:

$$dA2 = W3^T.dZ3$$

$$dZ2 = dA2 * g2'(Z2)$$

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

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

Did you notice a pattern here? For forward propagation, the input to the second layer (A2) is A1 which is the output of the first layer. To generalize it, we can say that input to the L

layer is the output of the L-1 layer or previous layer, denoted by A(L-1). So, the generalized form is:

$$Z(L) = W(L).A(L-1) + b(L)$$

$$A(L) = g(Z(L))$$

Where WL and bL are the weights and biases of the L layer, respectively, while A(L-1) is the output of the L-1 layer (a layer comes before the L-layer). Same goes for ZL and AL. One point to notice is that A(L-1) for the first layer, A1, is X (input data) and is denoted by A0 (A-zero).

The same intuition can be applied to backward propagation. For dA2 (second layer), we use the W3 and dZ3 (next layer terms). So, to generalize it, we can write:

$$dAL = W(L+1)^T.dZ(L+1)$$

$$dZL = dAL * gL'(ZL)$$

$$dWL = \frac{dZL.A(L-1)^T}{m}$$

$$dbL = \frac{\sum dZL}{m}$$

Where (L-1) means a layer before the L layer and (L+1) means a layer after the L layer.

Now let's explore some data preprocessing techniques. There are different ways to preprocess the data but I will discuss the two types only. They are:

Data Cleaning: This is simply removing missing or duplicate data, correcting errors, and removing outliers that can adversely affect the model's performance. We simply use filters and functions in Excel, and in Python, the NumPy library provides convenient functions like np.null, np.unique, etc. to clean our data.

Data Scaling: Machine learning algorithms often perform better when the input data are scaled to a similar range. For example, all the values of input are between zero and one or something like that. We usually do this by dividing all the input values by the maximum values of that input. For example, if X is [1,2,3,4,5], we will divide all the values by the maximum value of X (which is 5 in this case) to get [0.2,0.4,0.6,0.4, 1.0]. If we have hundreds of thousands of data, we simply use a max function in Excel and in Python to find the maximum value. Generally, this method is called min-max normalization and is used for positive data only.

We also use Z- score normalization. This means we transform the data so that it has a mean of zero and a standard deviation of one. This can be done by below formula:

$$\text{X\_normalized} = \frac{X - Mean\ of\ X}{\text{Standard Deviation of X}}$$

For example, for a data set X = [1,2,3,4,5], the mean is 3 and the standard deviation is √2 or 1.41. So, X_normalized for the first value of X, which is 1, is:

$$\text{X\_normalized} = \frac{1 - 3}{1.41} = -1.41$$

After normalizing all the values of X, we have X_normalized = [-1.41, 0.70, 0, 0.70, 1.41]. All data range from -1.41 to 1.41.

Now let's discuss the training and testing data sets. In machine learning, we use the training and testing data sets to train and evaluate our models. The training data set is used to train the model, while the testing data set is used to evaluate the model's performance. The idea behind using separate data sets is to check the performance of the model on unseen data or new data. If we use the same data set for training and testing, the model will perform well on that specific data set (training set), but it might not generalize well on new data. This is known as overfitting, and it can lead to poor performance when the model is used to make predictions on new data. Therefore, to avoid overfitting, we randomly split the data into training and testing data sets. The usual split is 80% of the data for training and 20% for testing, but this can vary depending on the size of the data set and the complexity of the model.

Incidentally, underfitting occurs when the model is too simple and cannot capture the underlying patterns in the data. In other words, the model does not fit the data well enough and performs poorly on both the training and testing data sets. To curb underfitting, we usually increase the model complexity (increasing the number of hidden layers and/or neurons), gather more data, etc.

Now it's time to combine all our knowledge and apply it to a real set of data. But first of all, I would like to thank Professor Michael Pyrcz of the University of Texas at Austin for generously providing me the porosity and permeability data, used in this article. The complete dataset is available here.

I am using the data of a file name "Stochastic_1D_por_perm_demo" in that repository. You can see all the Python code here. It is instructed to open it in a new window and read side by side of this article.

One caveat here is that it's important to note that porosity is just one of the factors that affects permeability. Therefore, it's important to keep in mind that our data presented in Table 1 below is for educational purposes only, as there are many other factors that affect permeability beyond just porosity.

	Unnamed: 0	Porosity	Permeability
0	0	13.746408	193.721529
1	1	9.608479	105.718666
2	2	11.664361	138.539297
3	3	8.375338	93.719985
4	4	13.183358	169.738824
100	100	8.042614	84.564471
101	101	19.887759	453.228801
102	102	11.118544	128.606149
103	103	16.051620	265.341789
104	104	17.762477	337.335817

Table 1

Table 1 displays the top and bottom 5 cells of our data. The numbering of the samples starts from zero, indicating that there are a total of 105 samples. Column 1 (Unnamed: 0) contains serial numbers, column 2 shows porosity, and column 3 displays permeability. We don't want serial numbers (column 1) to be a part of our data, right? Let's delete it using the drop function of Pandas. Additionally, I checked for missing values, and fortunately, our data does not have any null values. The updated version of the data is shown in Table 2 below.

	Porosity	Permeability
0	13.746408	193.721529
1	9.608479	105.718666
2	11.664361	138.539297
3	8.375338	93.719985
4	13.183358	169.738824
100	8.042614	84.564471
101	19.887759	453.228801
102	11.118544	128.606149
103	16.051620	265.341789
104	17.762477	337.335817

Table 2

Let us consider X as our input variable, which represents porosity, and Y as our output variable, which represents permeability. Figure 1 displays the curve between the two variables.

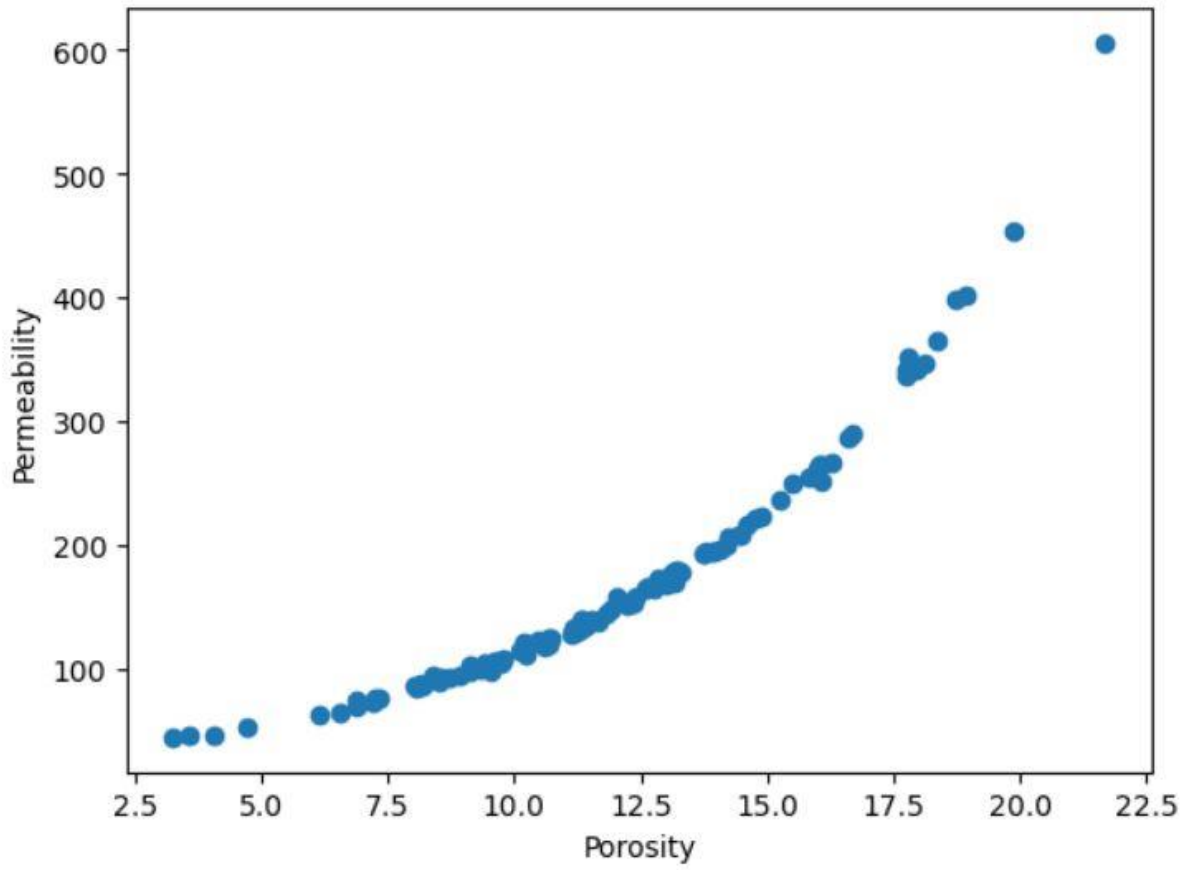

Figure 1: Porosity vs Permeability plot. All images created by author.

The x-axis of Figure 1 displays the porosity values ranging from 2.5 to 22.5. Let's scale it to range from 0 to 1 by dividing all the X values by the maximum values of X. We can accomplish this in Python using the command 'X = X/np.max(X)'. Once the scaling is done, we can redraw the curve between X and Y, and this time the x-axis will range from 0 to 1. The resulting graph is presented in Figure 2 below.

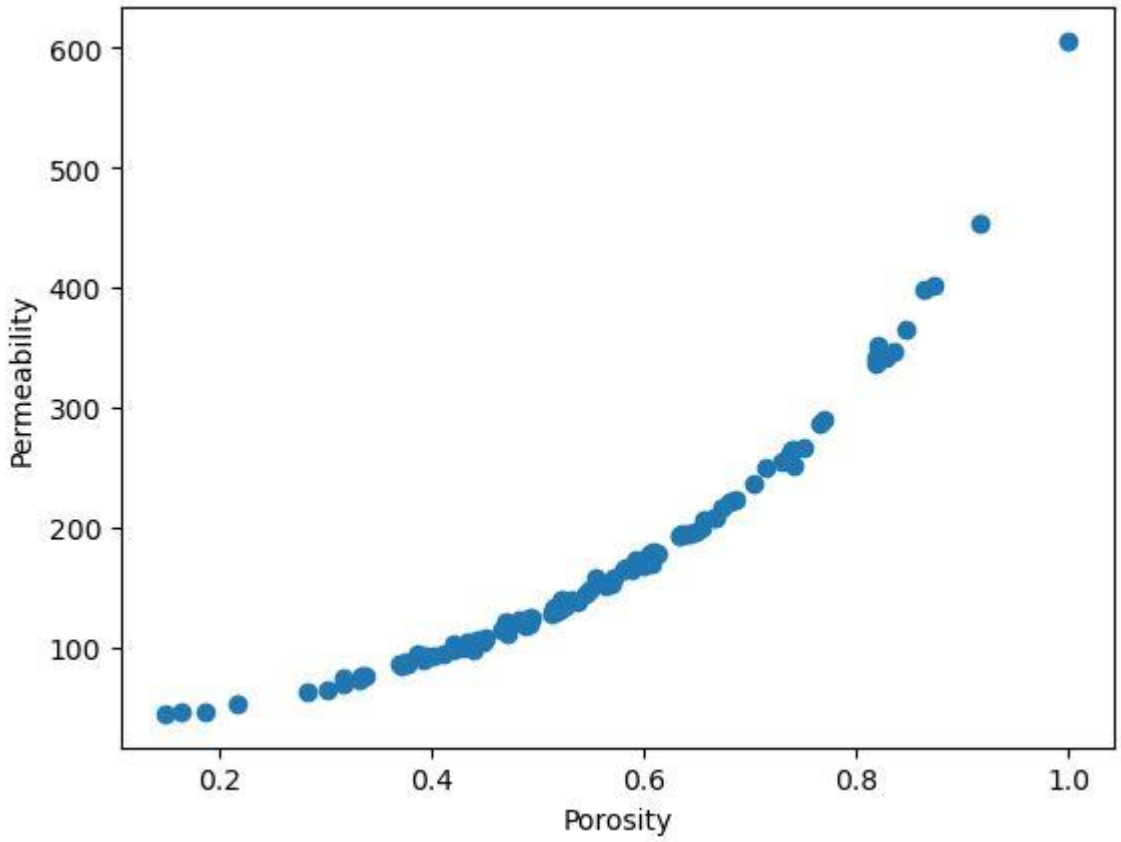

Figure 2: Porosity vs Permeability Scaled plot

Upon comparing Figure 1 and Figure 2, it is evident that they are similar except for the fact that in Figure 2, X values are scaled to range from 0 to 1. This scaling improves the performance of the model and reduces the running time. It is recommended to scale the input data to a suitable range to ensure that the model can effectively capture the underlying patterns and relationships between the input and output variables.

To prepare our data for training and testing, we use a function called **split_data** which splits our dataset into training and testing sets. The training set consists of 80% of our data, while the testing set consists of the remaining 20%. Additionally, this function transposes our data so that the features (porosity) are in a row, and the number of examples is in a column. As a result, we obtain the following dimensions for our data:

X_train shape: (1, 84)

Y_train shape: (1, 84)

X_test shape: (1, 21)

Y_test shape: (1, 21)

These dimensions indicate that we have 84 samples for training and 21 samples for testing.

The next function is **initialize_parameters_deep**. This function initializes the parameters (weights and biases) using the He et al. (2015) initialization method, as discussed in part 3. This method helps to improve the convergence of our neural network and prevent vanishing or exploding gradients.

The **linear_forward** function defines the linear function, which is represented as Z = WX + b or Z = WA + b. The **relu** function applies the ReLU activation function to the output of the linear function, which is represented as A = g(Z). The **linear_activation_forward** function combines both the linear_forward and relu functions to perform the forward propagation step of our neural network. Finally, the **compute_cost** function calculates the cost (J) of our model. Together, these functions make up the feedforward propagation step of our neural network.

Now it's time to implement the backward propagation step to update our parameters. To do this, we need to compute the gradients (derivatives) of the cost function with respect to the parameters. From part 3, we know that the derivative of the loss with respect to the output of the last layer, denoted by AL or $\hat{Y}$, is simply dAL = AL - Y, where Y is the actual value of permeability. This will be used as the starting point for the backward propagation algorithm. We feed this to ***relu_backward*** function to determine the derivative of Z with respect to the loss, denoted by dZ. Next, we use the **linear_backward** function to determine the derivatives of the weights (dW), biases (db), and the previous layer's activations (dA_prev) with respect to the loss. Finally, we use the **linear_activation_backward** function to call the relu_backward and linear_backward functions and compute the derivatives for the entire layer.

Furthermore, the **update_parameters** function is used to update the parameters. The **L_model_forward** function performs forward propagation through all L layers of the network, while the **L_model_backward** function performs backward propagation through all L layers of the network to compute the gradients. Finally, the **L_layer_model** function integrates all the aforementioned functions to train an L-layer neural network. With the completion of the implementation, we are now ready to run the model on our data.

We define the number of layers and neurons using **layers_dims**. Let's try different models. For Model 1, we assume that layers_dims is equal to [X_train.shape[0], 3, Y_train.shape[0]]. This means that the number of input features is 1 (porosity only) as indicated by X_train.shape[0]. The hidden layer has three neurons, and the output layer's neuron is 1 (permeability only) as indicated by Y_train.shape[0]. We set the learning rate to 0.01 and the number of iterations to 1000. By running our Model 1 with the L_layer_model function, we obtained the following results.

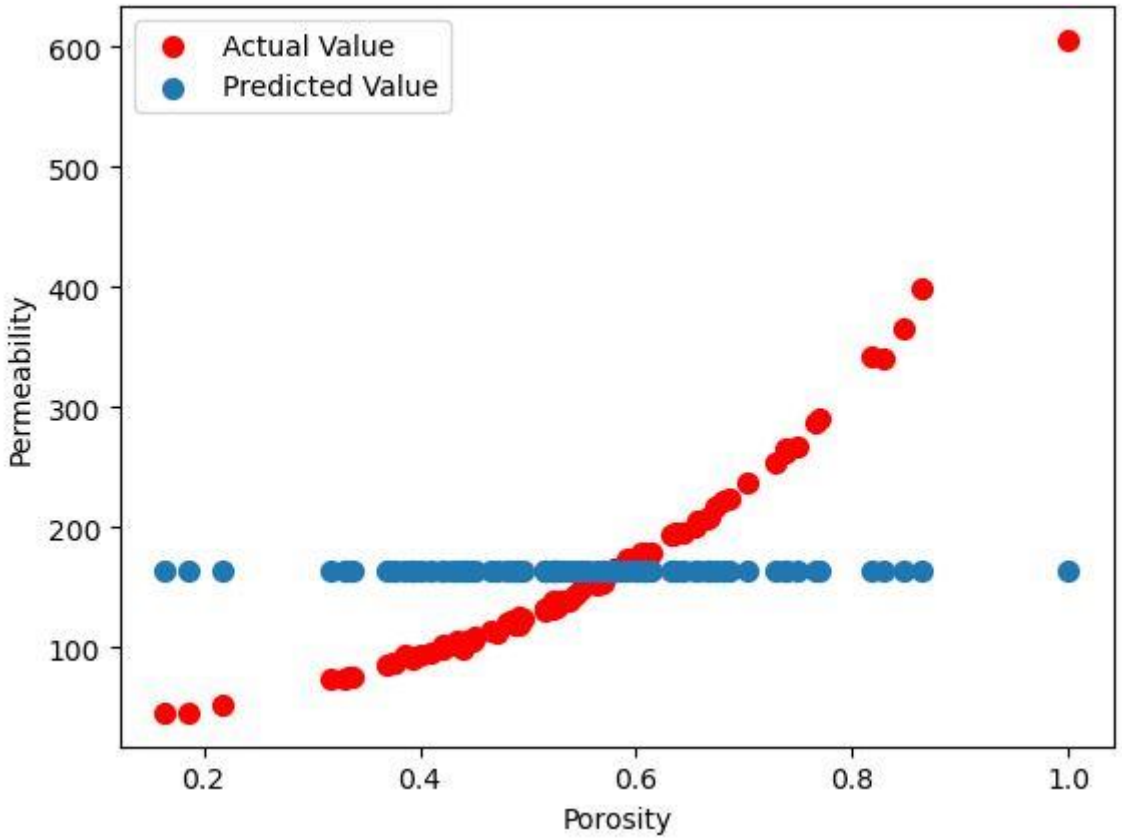

Figure 3: Model 1

You can see that the predicted values of permeability in model 1 are quite different from the actual values, indicating a poor fit.

Let's try model 2, where we set the layers_dims to [X_train.shape[0], 3, 5, Y_train.shape[0]]. This means that we have two hidden layers, with the first layer having three neurons and the second layer having five neurons. The number of input features and output neurons is the same as in the previous model. We will keep the learning rate and the number of iterations the same as in the previous model.

After running our model 2 using the L_layer_model function, we obtained the following results.

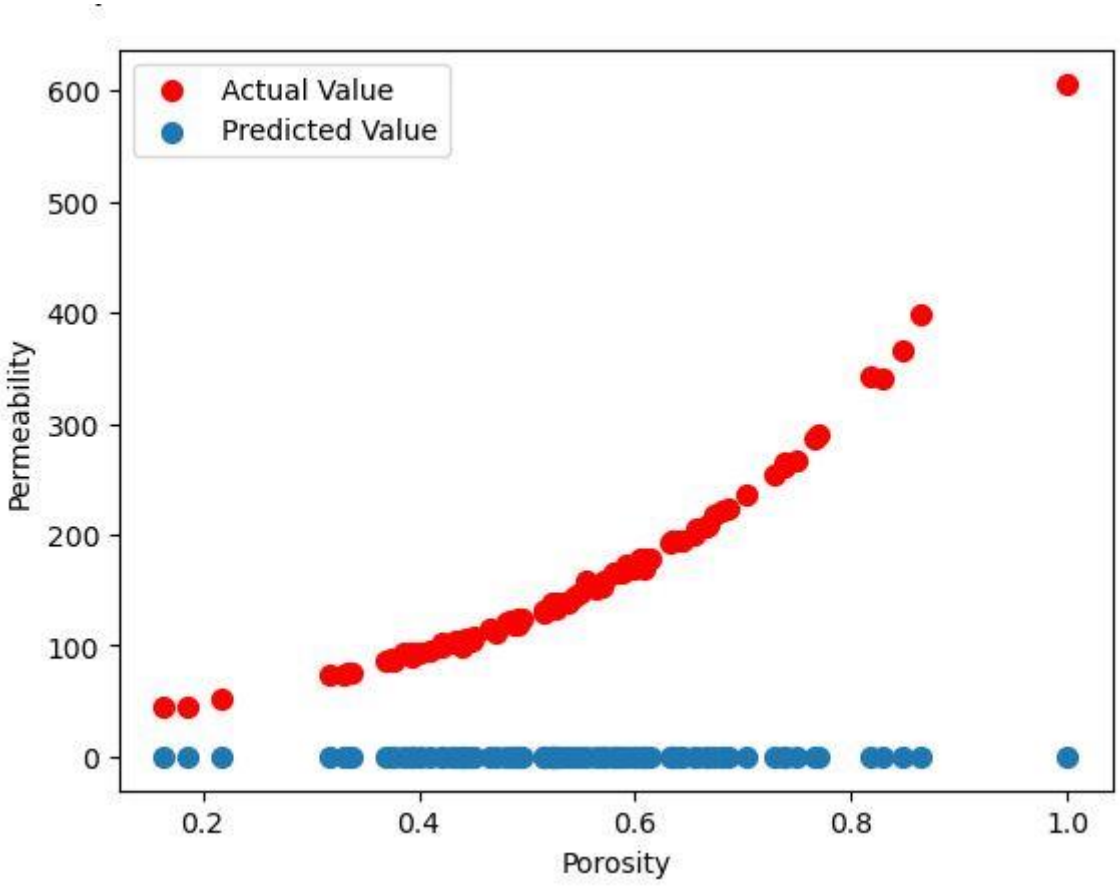

Figure 4: Model 2

Once again, we can observe a poor fit between the actual and predicted values of permeability for model 2. Let's try model 3 with the same number of hidden layers and iterations as in model 2, but the learning rate is decreased to 0.001. The Result of model 3 is shown in the below figure.

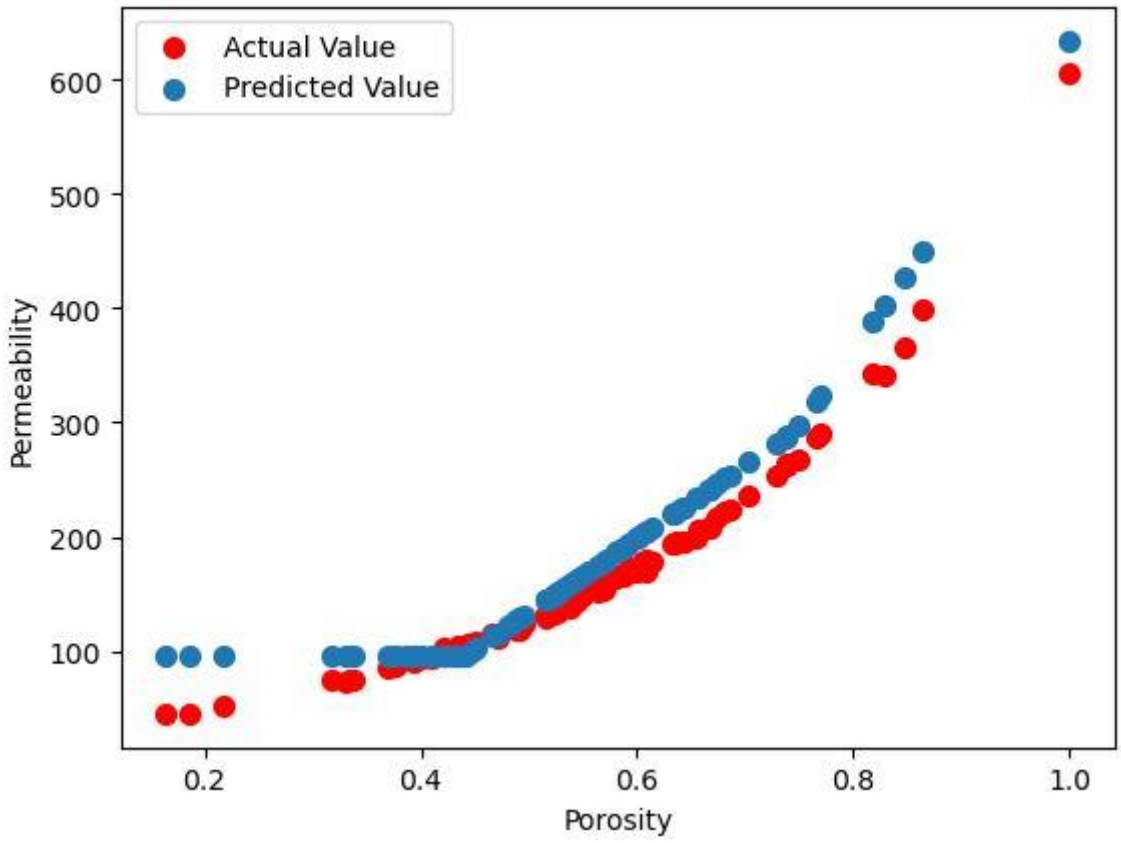

Figure 5: Model 3

The performance of model 3 is much better than the previous two, with a training accuracy of 85.37%. However, there is still room for improvement. We can try tweaking the hyperparameters, such as learning rate and number of iterations, or try adding more layers and neurons to see if the model performance improves.

For model 4, we increase the number of iterations to 5000 and decrease the learning rate to 0.0001. The results are shown in the figure below.

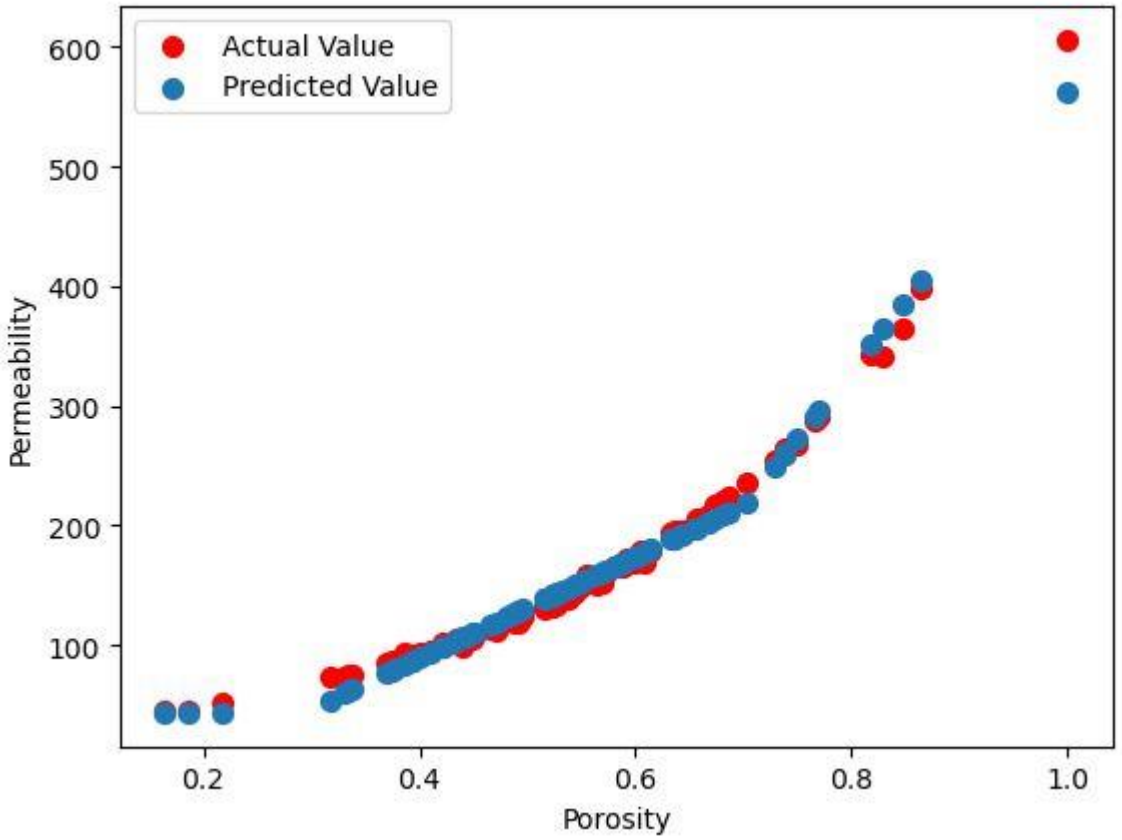

Figure 6: Model 4

Wow! Model 4 performed very well, with a training accuracy of 95.30% and a training error of 4.69%. However, the ultimate goal of our model is to make accurate predictions on new, unseen data. Let's evaluate the model's performance on our testing data.

The **test_model** function uses the parameters (W and b) learned by model 4 and predicts the permeability for the X_test values. The resulting predicted permeability values are

shown below.

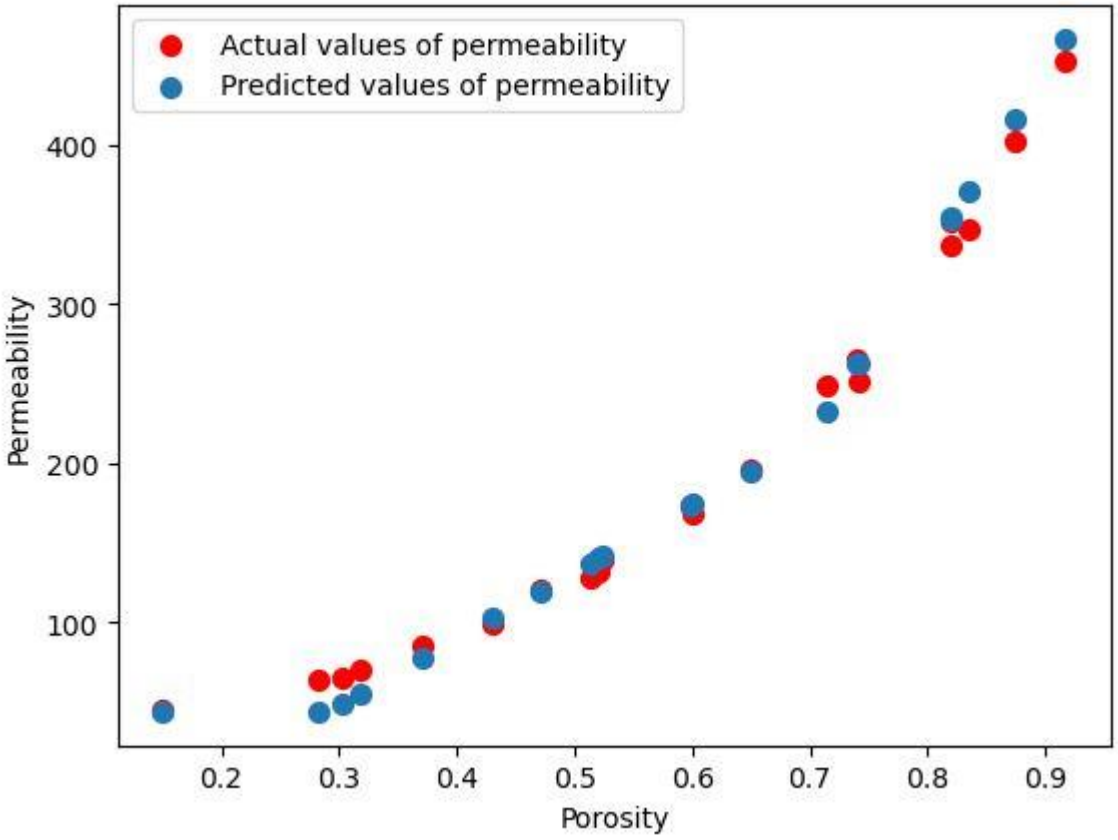

Figure 7: Test result

The testing accuracy of 93.11% and testing error of 6.88% indicate that our model generalized well and did not overfit. Below Table 3 shows the first five and bottom five values of actual and predicted permeability:

	Actual Permeability	Predicted Permeability
0	65.227602	48.423159
1	98.809947	102.649599
2	173.684221	173.418657
3	251.375148	263.172400
4	120.779829	119.351362
16	84.564471	77.245457
17	453.228801	466.938151
18	128.606149	137.328746
19	265.341789	262.251597
20	337.335817	353.538548

Table 3

We can see that the predicted permeability values are very close to the actual values for most of the test data, indicating the effectiveness of our neural network model.

If you wish to represent this model in mathematical notation, it can be written as follows:

$$Permeability = Relu(W3.Relu(W2.Relu(W1.Porosity + b1) + b2) + b3)$$

This equation represents a mathematical form of our neural network model 4 with three layers. The equation takes the porosity as an input and predicts the permeability as an output. The input feature, porosity, is multiplied by the weight matrix W1 and added to the bias vector b1. The resulting value is then passed through the ReLU activation function to introduce non-linearity. The output of the ReLU function is then multiplied by the weight matrix W2 and added to the bias vector b2. This process is repeated for the remaining two hidden layers (W3, b3). Finally, the output of the last ReLU function is the predicted permeability.

Figure 8 below displays the values of all parameters that were learned by model 4. These parameters include weights and biases for each layer in the neural network and these learned parameters are used to make predictions on new, unseen data.

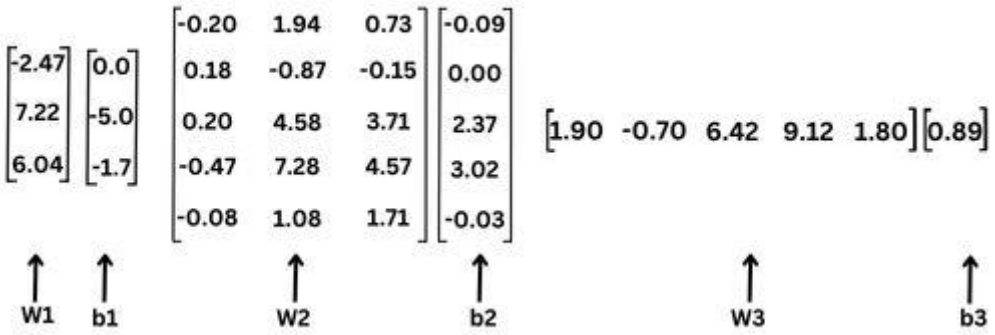

Figure 8: Values of parameters, learned by Model 4

So, that's the end of this article and the end of our machine learning journey. Thank you for reading until the end. Now that you have learned how to apply machine learning to oil and gas problems, it's important to keep in mind that the quality of your model depends on the quality of your data. As the saying goes, "garbage in, garbage out." Therefore, it's essential to ensure that your data is of high quality to obtain accurate and reliable results.

Feel free to customize the code provided to suit your own data and specific needs. I hope these four series of articles have been helpful to you and that you feel confident to apply machine learning techniques to your own oil and gas problems.

Once again, thank you for your time and attention. See you soon with more freshly brewed content!