Neural Net

Jacob Clostio

Nicholas Leopold

Abstract

In this paper, the approach, design, and implementation of a Neural Network classifier/model will be explored. A neural network, specifically a Multi-layer Perceptron with backpropagation, is a supervised learning model that utilizes basic units known as neurons organized into layers as well as weights and biases to help classify data. As a result of its ability to learn non-linear models, to expand horizontally and vertically with tunable parameters such as the number of hidden layers and the number of nodes, and to handle large amounts of input data accurately, the Multi-layer Perceptron is a popular algorithm to solve machine learning problems.

Hypothesis

Regardless of whether the model is using regression or classification data, we expect the rate of convergence, measured in the number of epochs needed to converge, will inversely correspond to the performance of the model on the test set. That is, the longer the model takes to converge, the worse the performance will be on the test set. The reasoning behind this hypothesis follows the logic that a longer convergence rate implies the minimum error in the model is difficult to find. This could be caused by local error minimums that detract the model from finding weights that globally minimize the error.

About a Multi-layer Perceptron

As stated in the abstract, a Multi-layer Perceptron is a feed forward supervised model that classifies data through the use of weight matrices, bias nodes, an activation function, and in our implementation, a back propagation algorithm to adjust weights and biases. This algorithm takes in the number of hidden layers, the number of hidden nodes per layer, and a numerical data set split into a test set and a training set. In our Neural network, we only need the test set when we test our final model since we implemented a training function and a test function. After furnishing these inputs to the Multi-layer Perceptron, it can begin to create the best model to fit the data. The first step is to create the input nodes. There will be the same amount of inputs as features in the training set, and the values inside the nodes will correspond to their respective features. Depending on how many hidden layers were specified (or the number of nodes inside each layer), the neural net then creates bias nodes initialized to 0 for the first iteration, as well as weight matrices with random values x, where $-0.5 < x < 0.5$. The shape of the weight matrices corresponds to the left and right layers that the weight matrix connects. Finally, a label array with the size of the number of classes in the data set is created for classification. It will contain values of all 0's and one 1, where the 1 represents the correct class for that instance of training data. For regression, the output will be a single node comprised of the sum of the previous layer multiplied by the weights between the output and the previous layer. After all of these pieces have been put together in the Multi-layer Perceptron, it can now calculate the hidden node values and output based on the initialized weights and input. For the node values, there needs to be an activation function to determine whether a neuron will be activated or not. For classification problems we use the Sigmoid SoftMax function, and for regression we use a linear activation function. At the end of the first iteration, the output will most likely be a poor guess and classify the data incorrectly, which is why the weight matrices, bias nodes, all of the node values, the learning rate, and the momentum parameter are sent to the backpropagation algorithm. The backpropagation algorithm uses the output, label, and learning rate to determine the change (δ) needed to be added to both the weight matrices and bias nodes. This δ is easy to find between the last hidden layer and the output layer (simply output - label); however, for any other two layers, we need to take the derivative of our activation function and multiply it with our δ and then matrix multiply that with the corresponding weight matrix to find the update values for the weight matrix and bias nodes. This δ is effectively the gradient descent that will help train the model. The backpropagation algorithm also has the feature of momentum, which keeps track of the previous δ and if enabled will factor in the previous δ when calculating the new δ. This will help the algorithm avoid getting stuck in local minimums. Once all of the changes have been made to the weight matrix and bias nodes, it runs the forward propagation again with the updated weights. This repeats for the entire training set, and if the accuracy / MSE of the Multi-layer Perceptron is not good enough / has not converged based on a set threshold, the entire model can be trained again up to any amount of epochs. Once the model has converged to an acceptable accuracy, we can then use the test set on the finished model to see how it classifies results.

Design

The program implementation of our Multi-layer Perceptron Algorithm will be tested by using 10-fold cross validation, where the model is trained on a randomly selected portion of 90% of the data. Then the data is shuffled and a new randomly selected portion of 90% of the data is selected for training and the other 10% is left for testing. The algorithm will do this ten times. This will allow for a more robust MLP model.

Data Preprocessing

Before we run our algorithms on the data sets, we must clean the data. For the non-regression data sets, much of the cleaning is removing columns that are categorical, replacing missing values (Cancer data), and ensuring that values are in the correct type to do distance measurements (Soybean classes were encoded to be numerical). As for the regression data sets, the abalone data needs to be one-hot encoded to remove the nominal values 'M' (male), 'F' (Female), and 'I' (Infant). So simply replacing them with 0, 1, 2 would work. For the Computer hardware data set, a few categorical columns need to be dropped (Vendor name, Model name), and the target column needs to be moved to the final position. Finally, the forest fire data set has a few cyclical categorical values (days and months), this can be easily cleaned to be integer values of their respective day / month.

Algorithm

Our algorithm utilizes many Numpy matrices to keep track of all the weights and node values. Much of the data can only be stored in Numpy arrays, as the back propagation algorithm has matrix multiplication that requires the shapes of the matrices to match (columns of first to rows of second) to be able to work. The time complexity of the Multi-layer Perceptron depends on how how many epochs are necessary to train the model to be as accurate as we want it to be (or until it converges). For something like the cancer data set, that does not need many epochs to return a well trained accurate model, the time complexity would be close to $O(n)$. However, another data set that does need many epochs to return a well trained accurate would be closer to $O(n^2)$ or larger depending on when it converges.

Parameter Tuning

There are many tunable parameters in the implementation of a neural net. Of these parameters, the number of hidden layers, the number of nodes per hidden layer, the learning rate, and the momentum factor of the model were mainly used for tuning our data sets. The tables below entail the tuning process of each of our data sets. The values marked ’DNC’ denote accuracies or mean squared errors that did not converge to the desired threshold, indicating a poor performance based on the threshold.

Soybean Tuning

Layers	Nodes	Learning Rate	Momentum	Training Accuracy	Test Accuracy	Loss	Epochs
0	0	0.0008	0.5	76.74	DNC	DNC	DNC
0	0	0.000625	0.5	79.07	DNC	DNC	DNC
0	0	0.000625	0.2	79.07	DNC	DNC	DNC
0	0	0.000625	0	76.74	DNC	DNC	DNC
1	30	0.000625	0.5	90.69	100	0	171
1	30	0.000625	0.9	58.13	DNC	DNC	DNC
1	15	0.000625	0.5	90.69	50	2	562
2	[18,3]	0.000625	0.5	79.07	DNC	DNC	DNC
2	[10,6]	0.000625	0.5	90.69	50	2	3709
0	0	0.00625	0.5	90.69	75	1	54
0	0	0.01	0.5	93.02	0	4	11
0	0	0.01	0	93.02	25	3	11
0	0	0.01	0.9	93.02	75	1	13
0	0	0.001	0.25	79.07	DNC	DNC	DNC
0	0	0.00625	0.25	93.02	50	2	20
2	[30,2]	0.00625	0.5	30.32	DNC	DNC	DNC

z

Glass Tuning

Layers	Nodes	Learning Rate	Momentum	Training Accuracy	Test Accuracy	Loss	Epochs
2	[5,3]	0.0008	0.5	75.13	DNC	DNC	DNC
1	6	0.0008	0.5	76.15	DNC	DNC	DNC
0	0	0.0008	0.5	68.39	DNC	DNC	DNC
1	6	0.0006	0.5	76.68	DNC	DNC	DNC
1	6	0.01	0.5	73.35	DNC	DNC	DNC
1	6	0.0015	0.5	75.13	DNC	DNC	DNC
1	6	0.015	0.9	63.21	DNC	DNC	DNC
1	6	0.0006	0.9	46.11	DNC	DNC	DNC
1	6	0.0006	0	77.2	DNC	DNC	DNC
1	6	0.00065	0	86.01	66.66	7	4112
1	6	0.0007	0	82.9	DNC	DNC	DNC
1	6	0.0005	0	78.75	DNC	DNC	DNC
1	6	0.000675	0	78.75	DNC	DNC	DNC
1	6	0.000625	0	80.31	DNC	DNC	DNC

Breast Cancer Tuning							
Layers	Nodes	Learning Rate	Momentum	Training Accuracy	Test Accuracy	Loss	Epochs
2	[7,3]	0.000625	0.9	65.07	DNC	DNC	DNC
2	[7,3]	0.000625	0.5	96.5	97.1	2	137
2	[7,3]	0.000625	0.2	96.82	98.55	1	133
2	[7,3]	0.000625	0	96.19	100	0	67
1	8	0.000625	0	97.14	94.2	4	92
0	0	0.000625	0	95.18	92.75	5	107
2	[7,3]	0.0008	0	96.03	98.55	1	81
2	[7,3]	0.001	0	96.5	98.55	1	47
2	[7,3]	0.01	0	97.46	95.63	5	8

Abalone Tuning						
Layers	Nodes	Learning Rate	Momentum	Training MSE	Test MSE	Epochs
2	[5,2]	0.000625	0.5	3.12e-6	0.00014513	1
2	[5,2]	0.000625	0.5	3.12e-6	0.00014513	1
2	[5,2]	0.000625	0.5	3.10e-6	0.00014461	1
1	[6]	0.000625	0.5	3.31e-6	0.00014296	1
1	[6]	0.001	0.5	2.95e-6	0.00014237	1
1	[6]	0.01	0.5	3.40e-6	0.00014785	1
0	0	0.001	0.5	0.2428	DNC	DNC
0	0	0.001	0.5	0.02130211	0.01966948	1
1	[6]	0.001	0.9	3.20e-6	0.00015543	1
1	[6]	0.001	0	3.05e-6	0.00014685	1

Computer Tuning						
Layers	Nodes	Learning Rate	Momentum	Training MSE	Test MSE	Epochs
1	6	0.000625	0.5	1.15e-6	8.32e-6	1
2	[5, 2]	0.000625	0.5	1.82e-6	1.90e-5	1
0	[]	0.000625	0.5	DNC	DNC	DNC

Fire Tuning						
Layers	Nodes	Learning Rate	Momentum	Training MSE	Test MSE	Epochs
1	[10]	0.001	0.5	4.13e-6	5.82e-5	1
2	[5,2]	0.001	0.5	2.14e-6	4.83e-6	1
2	[5,2]	0.001	0.9	2.23e-6	4.14e-6	1
2	[5,2]	0.001	0	2.18e-6	3.69e-6	1
0	0	0.001	0	DNC	DNC	DNC
2	[5,2]	0.01	0	2.54e-6	1.67e-5	1
2	[5,2]	0.0001	0	3.01e-6	1.43e-5	1

From the tables, the most optimal parameters can be chosen based on the performance of the model on the test set. For the soybean data, the best parameters were 1 hidden layer with 30 nodes, a learning rate of 0.000625, and a momentum of 0.5. This resulted in

100% accuracy on the test set. The glass data training accuracy only converged past the threshold (a training set accuracy of 0.85) for one combination of parameters: 1 layer, 6 nodes, a learning rate of 0.00065 and no momentum. Although the glass data did eventually converge to the 0.85 benchmark, it took a considerable amount of epochs (4112) to do so. The breast cancer data reached 100% accuracy on the test set when there were 2 hidden layers, with 7 and 3 nodes respectively, a learning rate of 0.000625 with no momentum. For the regression data sets, the number of epochs needed to converge to the mean squared error threshold of 0.001 reduced significantly when comparing to the number of epochs needed for the classification data. The abalone data had the least mean squared error (MSE) when there was only 1 hidden layer with 6 nodes, a learning rate of 0.001, and a momentum of 0.5. Similarly, the computer hardware data also performed best with 1 hidden layer with 6 nodes, but with a learning rate of 0.000625. Unlike the other regression data sets, the fire data set performed best with 2 hidden layers comprising of 5 and 2 hidden nodes respectively. A learning rate of 0.001 and no momentum led to a MSE of 3.69e-6.

Results

The model trained with classification data did not necessarily improve in accuracy or 0-1 loss when the number of epochs was small compared to the other convergence rates. Although some low convergence rates corresponded with high accuracy, as in the case where the model performed with 100% accuracy after 67 epochs on the breast cancer data, not all low convergence rates had this same behavior. When the breast cancer model converged after just 8 epochs, the test accuracy was 95.63%, lower than the models that converged after 137 and 133 epochs. The soybean data also provided evidence against our hypothesis, with the highest test accuracy corresponding to the second highest number of epochs. The model trained with regression data surprisingly converged after 1 epoch each time the test was run. Different, albeit close mean squared errors resulted after tuning the parameters, but the convergence rate remained the same. Therefore, we have evidence against our hypothesis that a lower number of epochs (a higher convergence rate) corresponds to a model with better performance.

Summary

The Multi-layer Perceptron algorithm (with back propagation) is an effective algorithm for classifying data. With its ability to learn non-linear models, to expand horizontally and vertically with tunable parameters, and to handle large amounts of input data accurately, the algorithm can be used to solve a plethora of machine learning problems. Although there a 3666re benefits to using Neural Networks, it also requires a great deal of parameter tuning and this algorithm in particular requires conditionally independent data. In all, Neural Networks, specifically a Multi-layer Perceptron, is a good intermediate algorithm to show how models can be fit to nice data as well as data that is nonlinear and complex.