Project 4: Error Backpropagation Algorithm

Nazanin Yari
Johns Hopkins University
Introduction to Machine Learning - Section 8VL

Abstract

In this project, the effects of using backpropagation algorithms for classification purposes are studied in detail. The results of the experiments with and without a hidden layer are compared. The analysis of the algorithm over only one dataset showed that the classification accuracy has been improved using one hidden layer compared to the experiments done using no hidden layer and two hidden layers in the neural network.

Problem Statement

The purpose of this project is to develop the error backpropagation algorithm in the classical fee-forward artificial neural network. There are two main hyperparameters that control the architecture of a neural network: the number of hidden layers, and the number of nodes or neurons in each layer.[1] The most reliable way to select these parameters is through iteratively tuning the configuration during training to prevent overfitting or underfitting. The hypothesis is that selecting the number of neurons between the size of the input layer and the size of the output layer would be a good starting point. Choosing 1 hidden layer rather than 2 might prevent overfitting as well. Overfitting happens when too many neurons and layers are used for the limited amount of information contained in the training set. Implementing the network with no hidden layer might cause other problems such as underfitting the model.

Algorithm Implementations

Generally, there is some disagreement about how to count the number of hidden layers. The disagreement is about whether or not to count the input layer. For this project, the number of hidden layers excludes the input layer. Therefore, a multi-layer perceptron has an input layer, a user defined number of hidden layers, and one output layer.

For this assignment, the backpropagation algorithm is used for classification. The backpropagation algorithm is a supervised learning method for training the weights in a multi-layer feed-forward neural network. Feed-forward neural networks process the information of the neuron cells we input and produce an output. The output goes through each hidden layer, and finally the output layer. The back-prop algorithm then goes back into the network and updates the weights iteratively until the function approximates the classification with the desired accuracy or when it reaches the user predefined number of epochs. The network for this project is trained using stochastic gradient descent which involves a predefined number of iterations of exposing a training dataset to the network, forward propagating the inputs for each row, and backpropagating the error and updating the network weights. For this project, the number of output nodes is set to 1. The model implemented for this assignment only classifies the data with 2 classes.

Experimental Approach

Initialize Network. The initial weights have been randomly selected using the function `weight_network()`. This function takes input size, number of hidden layers and number of hidden nodes as its parameters. It then returns a list with elements of type dictionary, each associated to a layer

[1] "How to Configure the Number of Layers and Nodes in a ...." 27 Jul. 2018, https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/. Accessed 8 Dec. 2020.

in the network. The key of each dictionary is the layer name and the value is the randomly selected numbers with an additional row for bias.

Forward Propagation. The function `forward_propogate()` calculates an output by propagating an input through each layer of the neural network until the output layer nodes have their values. This function calls another function `weighted_sum()` to calculate the neuron activation using the initial weights. It uses the training dataset as its input for the hidden layer, and the output of the hidden layer as an input to calculate the output nodes values. Once the outputs are computed, they are transferred using sigmoid function. This function takes any input value and produces an output number between 0 and 1.

Back Propagation. The goal of using this algorithm is to create a model which minimizes the stochastic gradient descent error with respect to weights. The sum of squared error has been used for this project. The `backpropagate_errors()` function first calculates the error for the output neuron. The error is then used as an input to propagate backwards through the network to calculate the errors for the nodes in the hidden layers.

Updating Weights. Once the errors are calculated for each node, they are used to update the weights using `update_weight()` function. The function takes the previous weights, the output values, the errors calculated by the backpropagation, a user defined learning rate, and an input row.

Training Network. The function `train_network()` takes the initial weight network, the training dataset, and epoch number. It loops through each row of the training dataset, calculates the output values, it then compares the output value of the neuron in the output layer and compares it to the target and computes the sum of squared errors. Finally, it calculates the delta and updates the weights for the next row. When the function reads the whole dataset, it goes through another epoch until it reaches the number of epochs defined by the user. The function computes the error for each epoch and prints it out on the console. It also plots the epoch error rate. The function returns the updated weights calculated in the last epoch to be used for prediction.

Prediction. The final step of the algorithm is to make predictions using the trained neural network. The function named `predict()` implements this step. It takes the updated weights from the trained network, and the test dataset. It iterates through the dataset to calculate the output value for each row using the trained network weights and compares it to the predicted class. If the calculated output was greater than or equal to 0.5, the input row is classified as class 1, and if the output value is less than 0.5, it is classified as class 0. The function returns both predicted class array and the prediction error.

Preprocessing Dataset

Missing values. For the breast cancer dataset, the missing values have been found only in columns 7 or Bare Nuclei which have been replaced with the most commonly occurring value i.e. mode of that column. No missing values have been found in the glass dataset.

Class Numbers. The class values in the breast cancer dataset have been changed to 0 and 1 from 2 and 4 respectively. Regarding the glass dataset with 7 classes, the numbers have been changed to 0 through 6.

Standardizing Dataset. The values for both dataset have been normalized with mean 0 and standard deviation of 1. The function `standardized_data()` in the read_data_p4.py file has been used for this procedure.

Test_Train_Validation. In order to tune the hyperparameters, the dataset has been divided into three sections. A block of 60% of the data is the training set used to build the network, the block of 30% of the data is the validation set used to test different hyperparameters, and the remaining block of 10% of the data is the test set used to compare the performance of the neural network when using no hidden layer, 1 hidden layer and 2 hidden layers. The function `data_train_validation_test()` creates these three blocks by first shuffling the dataset and then calling `train_test_split()` function twice.

Data statistics. The mean and covariance of the cancer dataset is shown below. Since the glass dataset has not been used for model prediction, the statistics have not been shown here.

Class 0 _ Covariance	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
Clump Thickness	2.80	0.44	0.51	0.45	0.27	0.22	0.23	0.39	-0.03
Uniformity of Cell Size	0.44	0.82	0.64	0.34	0.39	0.44	0.30	0.55	0.02
Uniformity of Cell Shape	0.51	0.64	1.00	0.31	0.36	0.38	0.26	0.48	0.00
Marginal Adhesion	0.45	0.34	0.31	0.99	0.35	0.38	0.21	0.41	0.03
Single Epithelial Cell Size	0.27	0.39	0.36	0.35	0.84	0.33	0.20	0.50	-0.01
Bare Nuclei	0.22	0.44	0.38	0.38	0.33	1.35	0.24	0.33	0.07
Bland Chromatin	0.23	0.30	0.26	0.21	0.20	0.24	1.17	0.44	-0.02
Normal Nucleoli	0.39	0.55	0.48	0.41	0.50	0.33	0.44	1.12	0.03
Mitoses	-0.03	0.02	0.00	0.03	-0.01	0.07	-0.02	0.03	0.25

Class 1 _ Covariance	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
Clump Thickness	5.90	0.64	0.70	-1.14	0.08	-0.32	-0.10	-0.10	0.72
Uniformity of Cell Size	0.64	7.40	5.03	2.79	3.06	-0.30	2.39	2.76	1.68
Uniformity of Cell Shape	0.70	5.03	6.56	2.18	2.39	0.42	1.96	2.69	1.37
Marginal Adhesion	-1.14	2.79	2.18	10.31	1.62	2.16	2.42	1.94	1.70
Single Epithelial Cell Size	0.08	3.06	2.39	1.62	6.01	-0.06	1.18	1.86	2.11
Bare Nuclei	-0.32	-0.30	0.42	2.16	-0.06	10.00	0.94	-0.90	-0.21
Bland Chromatin	-0.10	2.39	1.96	2.42	1.18	0.94	5.17	1.91	0.34
Normal Nucleoli	-0.10	2.76	2.69	1.94	1.86	-0.90	1.91	11.23	1.89
Mitoses	0.72	1.68	1.37	1.70	2.11	-0.21	0.34	1.89	6.54

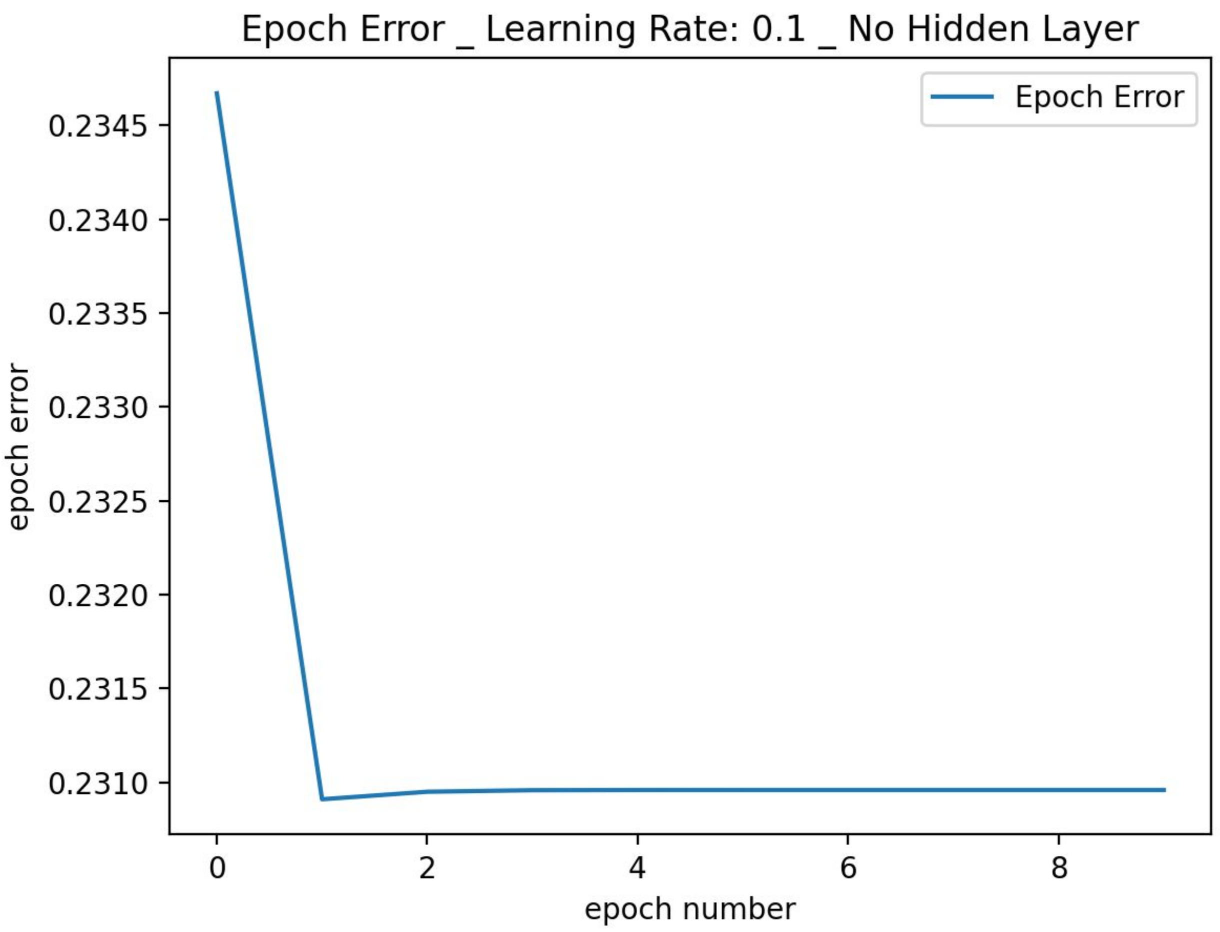
Mean	**Clump Thickness**	**Uniformity of Cell Size**	**Uniformity of Cell Shape**	**Marginal Adhesion**	**Single Epithelial Cell Size**	**Bare Nuclei**	**Bland Chromatin**	**Normal Nucleoli**	**Mitoses**
Class 0	2.96	1.33	1.44	1.36	2.12	1.34	2.10	1.29	1.06
Class 1	7.20	6.57	6.56	5.55	5.30	7.57	5.98	5.86	2.59

Results

The validation dataset has been used to set the hyperparameters; the number of hidden layers, and the number of hidden nodes in each layer as well as the learning rate and the number of epochs used to train the dataset.

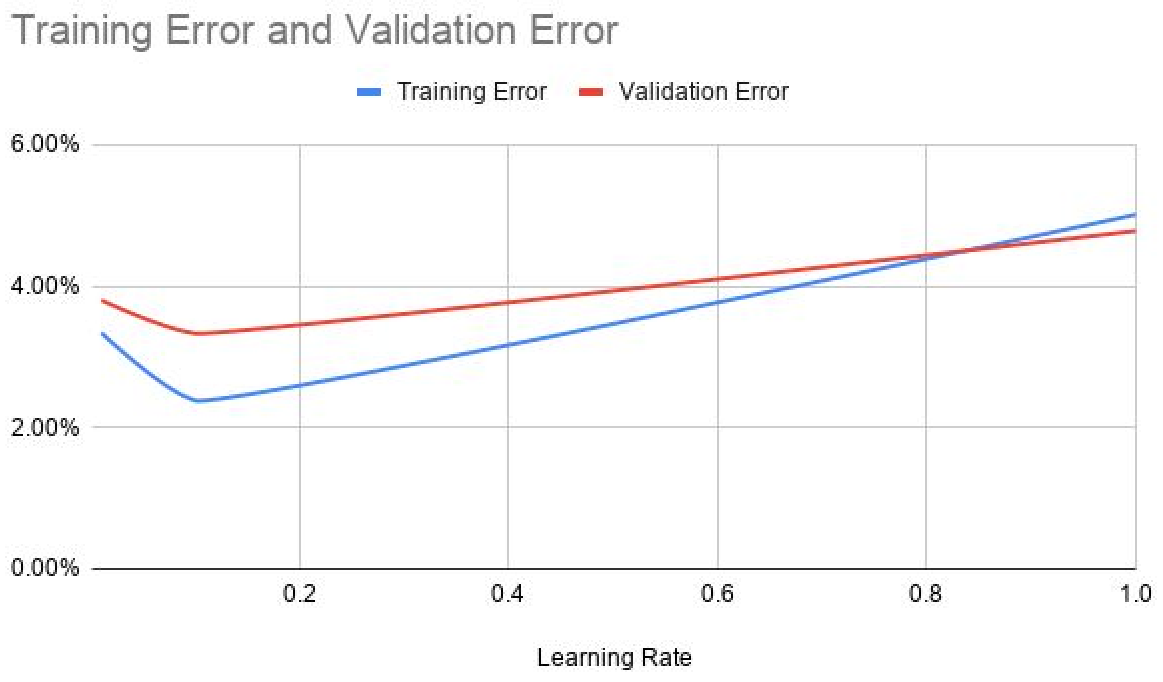
Learning rate controls how much the user wants to change the weight to reduce the error. Small learning rates result in slower learning over a large number of iterations. This prevents the network from converging prematurely by increasing the likelihood of finding a good set of weights that minimize the error.[2]

No Hidden Layer: The training dataset error with no hidden layer was 35.8%. The validation dataset has been used in order to set the learning parameter and the number of epochs needed to train the dataset. The below plot depicts that an epoch number of 2 was sufficient to train the dataset since the epoch error has not been changed after that. The learning rate used to train the dataset was 0.1. The error rate on the test dataset was 30.48%. It appears that using no hidden layer resulted in an underfit model.



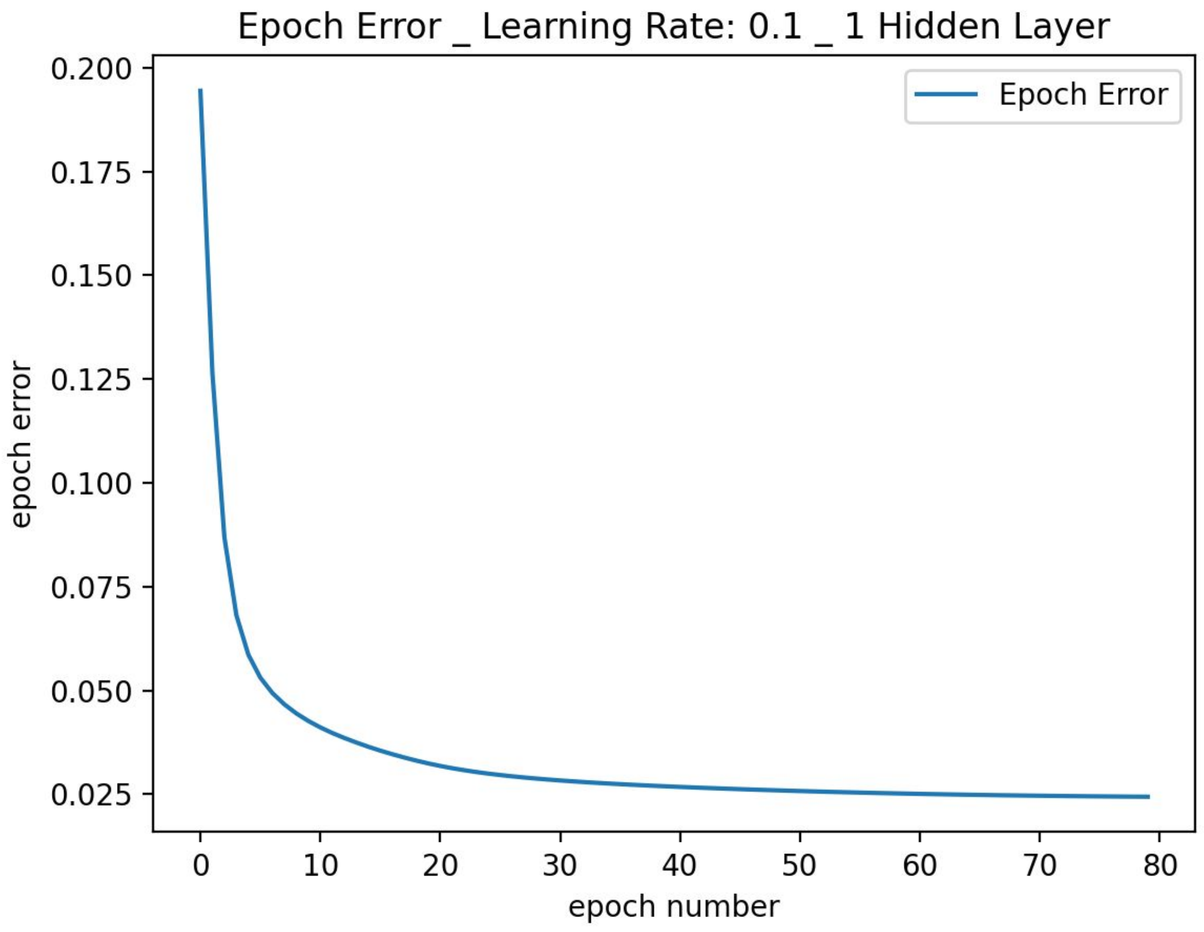
[2] "How to Code a Neural Network with Backpropagation In Python." 7 Nov. 2016, https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/. Accessed 10 Dec. 2020.

One Hidden Layer: Selecting the number of neurons between the size of the input layer and the size of the output layer is suggested. Using 5 hidden nodes, different learning errors have been tested on the validation set. For the small learning rates, the epoch number has been increased to correctly predict the error. As depicted in the below graph, the validation error rate with a learning rate of 0.1 is the lowest rate.



It seems that the network with a learning rate of 0.01 with 160 epochs has not been trained well enough. Increasing the number of epochs took significantly longer to train the network. The epoch error rate for the learning rate of 0.1 is shown below. For this learning rate, an epoch number of 20 was sufficient to train the network.

Please note that the number of hidden nodes smaller and larger than 5 have also been tested which resulted in higher validation error rate. Testing the updated weights from the network trained using the learning rate of 0.1 on the test dataset resulted in an error rate of 2.86%.



Two Hidden Layers: For the case with two hidden layers, the learning rate has been set to 0.1. The validation set has been used to test the number of nodes that should be used in each layer to train the model. Using 5 nodes in each layer resulted in a training rate error of 5.96% and validation error of 6.66%. Using only 1 node in each layer underfit the model and resulted in a very high training rate error of 35.79% and a validation error rate of 30.47%. However, when 15 nodes have been used in each layer of the network, the validation error was not as high as expected. The hypothesis was that increasing the capacity of the network would result in a good training rate error but a poor prediction due to overfitting the model. Using the cancer dataset, we could not support the hypothesis. I would like to test the network on a larger dataset in the future to see the impact of overfitting the model using a large number of nodes in hidden layers. The error rate of the classification on the test dataset with 5 nodes in each layer and learning rate of 0.1 was 7.1%.

Discussion

The analysis of the algorithm over the cancer dataset with 699 data points showed a better classification accuracy rate when one hidden layer with 5 neurons has been used. This result was expected since most using 1 hidden layer for 9 features would prevent overfitting or underfitting the model.

Summery

The network which has been trained with one hidden layer and the number of nodes between the size of the input layer and the size of the output layer with a learning rate of 0.1 had the lowest prediction error rate.