Neural Network

Ian Kessler ian.kessler@student.montana.edu
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Ethan Skelton ethan.skelton@student.montana.edu
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA

Editor: Ian Kessler and Ethan Skelton

Abstract

In this project, we implemented a feed forward neural network in order to train a model and try and predict the outcome from given features. We are using a back propagation and gradient decent in order to tune the weights matrices to predict the value. Then we applied 10 fold cross validation to our training sets to test on along with tuning for the number of hidden layers that was best for each data set. Our data sets comprise of breast cancer data, glass data, hardware data, abalone data, soybeans data, and forest fires data which each have there defined labels for their features along with target values associated with each feature.

Half of our data was classification based where we had classes to compare the data to. These data sets were breastcancer, glass, and soy beans, where the other classes are regression based: forest fire, abalone, and hardware. For the regression based models had to deal with the data in a bit of a different way. The output of the regression models only had one output where in the classification models the output number was the amount of classes the data set had.For any categorical values that we had we turned them into quantitative values so that we can use them for calculating hidden layers and outputs. For this we used one hod coding.

Keywords: neural network, back propagation

1. Problem Statement

For any of the size given data sets, we could train a model to predict the class of any example from the respective data set. We are trying to predict breast cancer, type of glass, type of soybean, age of abalone, performance of hardware, and area of a fire.

Our model is based off of back propagating feed forward neural network in order to predict these values. For a given amount of hidden layers our network will try and adjust weight matrices using gradient decent to give us the correct output for a given data set. We will tune the number of hidden nodes in each layer so that we can get the best output that we can. Along with that we are using a linear activation function to determine the output for the regression data sets and a soft max activation function to determine the class for the classification data sets.

We hypothesize that a smaller learning rate will be better. With a larger learning rate, we hypothesize that our change in weights will result in jumping too far from the optimal weights, but with a smaller learning rate, we can make smaller direction changes to easily "land" at the best weights.

We hypothesize that implementing momentum will increase performance. Averaging gradients should have an effect of having the weights change relative to the whole data set, which seems ideal because the samples collectively should have a similar pattern. Momentum seems especially optimal for stochastic online gradient descent, where we are updating weights randomly. It seems momentum would help decrease noise from the random changes in gradients.

We hypothesize that adding hidden layers will increase performance. Adding hidden layers give more dimensions and more control over accurately changing our weights to what is optimal for each data set. We hypothesize that two layers is better than one layer, since there will be even more dimensions to work with in converging to the optimal weights.

1.1 Feed Forward Neural Network

The feed forward neural network uses gradient decent in order to tune all of the weight matrices so that the network can predict a value from a class. It calculates each layer the same by adding up all of the nodes multiplied by their weights corresponding to the next layer. It will continue to do this all the way up to the output layer where regression models only have one output and classification models have as many outputs as there are classes.

2. Experimental Approach and Program Design

2.1 Data Preprocessing

2.1.1 Data Retrieval and Data Description

We downloaded the data sets as csv files from UCI Machine Learning repository at `https://archive.ics.uci.edu/ml/`. Some of these data sets did not have the attribute names or the class names labeled to the columns, and others did. We had to create a way to deal with both of these situations. For the situation where we did not have labeled columns we have a text file to know what columns names to append to each csv file. For the case where we already had the column names, we left the csv file as is.

2.1.2 Replace Missing Values

Some of the data sets had missing or unknown values denoted by '?'. For each of these data sets, we replaced each '?' with the same value that appeared frequently within the data set. In particular, in the "Breast Cancer" data set, we replaced each '?' with '3'. Luckily there was only the 'Breast Cancer' data set that was missing data, but if we needed to we could have changed out and other datasets '?' value with a value of our choice.

2.1.3 Categorical Values

There were some data sets that had categorical values that do not work well with the algorithms that we are trying to implement. We decided to deal with these values at the

beginning when we read the data set in. If we found values that were categorical then we used one hot coding in order to make them into quantitative values.

2.1.4 Data Sets

Half of our data sets were classification based: breastcancer, soybeans, and glass. For these data sets we defined that they were classification based which meant they had a column that was dedicated to the class that each data point belongs to. We used this column and any values in this column as the output layer of our feed forward neural network. The other half of our dataset was regression based: forest fires, hardware, and abalone. These datasets did not have a set of certain values that could be in the target row so there is only one output value in there neural nets. We used linear activation in order to run through the network and calculate the value that fit feature values best based on the weights of the throughout the network.

2.2 Experimental Design

2.2.1 10-Fold Cross Validation

To perform 10-Fold Cross Validation on a data set, we first partitioned the data set into 10 subsets of almost equal size. Using that partition, we tested our classification algorithm on each part of the partition by a model trained on the rest of the data.

2.2.2 Back Propagation

Our model back propagation in order to calculate all of the weight values throughout the neural net. We ran our neural net which created randomly generated weight values in between each layer. Which resulted in an output value. This output value was generally off so we use back propagation to go through each of the layers and adjust the weight vectors so that the output value would be as close as possible to the value that we expected to see.

2.2.3 Gradient Decent and Weight Matrices

In order to do back propagation correctly there has to be some reason to adjust certain weight values over others. This is where gradient decent comes in. It determines which values to change by a large amount and which values only need to change my a small amount. We continued to do this until we saw the performance of the algorithm stop improving. Which is where we would return our final weight matrices.

2.2.4 Tuning

For each training set fold, we tuned over η, which is the learning rate, α, which is the momentum weight, the number of hidden layers, and the number of nodes per hidden layer. For the learning rate, we tuned over $\{0.1, 0.2, 0.3\}$. For the momentum weight, we tuned over $\{0, 0.8, 0.9\}$. For the momentum weight, we tuned over $\{0, 0.8, 0.9\}$.

We had to tune the amount of hidden nodes in each of the hidden layers. In order to do this we would run the program on a certain hidden layer and then calculate the error. We would then add another hidden node onto it and repeat the process until we saw a decline

in performance. We tuned with zero layers, one layer, and two layers, and found the best results for each of those setups.

2.2.5 Sigmoid

The sigmoid function is used to squish certain numerical values down in between 0 and 1.

2.2.6 Linear Activation

We ran thought the neural net and calculated each of the layers and in the end the final layer had some weights associated with it that we multiplied the hidden nodes values by to determine the value of the data set.

2.2.7 Softmax

When we calculated the values of the output values we took the maximum value as the correct class.

3. Results

3.1 Data

We calculated error in our tuning process over each of the ten training sets. For the classification sets, we used cross-entropy error. For the regression sets, we used mean squared error. After we calculated error from the training sets, for each number of hidden layers, we found the best result from each fold and used those hyperparameters to test on their respective test sets. The following six tables are views of the first five rows of error from the respective data sets. The rest of the tables display the best hyperparameters to use for each fold according to their training set and the respective error found from the prediction on the test set.

3.1.1 Abalone

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	5.364186e+02	[3]	7.236470	[3, 3]	7.574557
1	0	0.1	0.8	2.729201e+02	[6]	6.654949	[6, 1]	7.076948
2	0	0.1	0.9	2.292186e+02	[4]	6.629368	[4, 3]	7.071675
3	0	0.2	0.0	4.552894e+150	[3]	7.862911	[3, 2]	8.127970
4	0	0.2	0.8	2.215516e+05	[3]	8.687298	[3, 5]	8.729821

Table 1: Abalone

3.1.2 Forest Fires

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	6.515480e+18	[1]	4648.059630	[1, 2]	4478.802056
1	0	0.1	0.8	2.811425e+16	[1]	4558.412481	[1, 2]	4547.017795
2	0	0.1	0.9	8.299379e+11	[1]	4645.714470	[1, 2]	4546.465444
3	0	0.2	0.0	6.580640e+73	[1]	4749.908839	[1, 1]	4694.809008
4	0	0.2	0.8	4.025594e+36	[1]	4719.231747	[1, 1]	4582.881148

Table 2: ForestFires

3.1.3 Hardware

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	2.050000e+20	[4]	20202.84928	[4, 1]	37400.53309
1	0	0.1	0.8	1.850000e+17	[2]	20040.64746	[2, 2]	26442.34421
2	0	0.1	0.9	6.552500e+14	[2]	23837.66255	[2, 1]	26487.21731
3	0	0.2	0.0	2.500000e+44	[1]	35734.37505	[1, 1]	31559.75976
4	0	0.2	0.8	1.270000e+26	[1]	37319.43115	[1, 1]	22806.62133

Table 3: Hardware

3.1.4 Breast Cancer

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	0.007275	[4]	0.004718	[4, 2]	0.060983
1	0	0.1	0.8	0.000003	[2]	0.044996	[2, 2]	0.046731
2	0	0.1	0.9	0.012003	[1]	0.696713	[1, 2]	0.022688
3	0	0.2	0.0	0.006812	[2]	0.004698	[2, 3]	0.006358
4	0	0.2	0.8	0.000102	[1]	0.071853	[1, 2]	0.048157

Table 4: BreastCancer

3.1.5 SoyBeans

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	0.001761	[1]	1.706611	[1, 3]	1.553919
1	0	0.1	0.8	5.801134	[3]	1.663139	[3, 2]	1.473419
2	0	0.1	0.9	4.920769	[2]	2.103138	[2, 1]	1.442491
3	0	0.2	0.0	8.211602	[1]	1.603580	[1, 3]	1.679193
4	0	0.2	0.8	10.560552	[2]	1.512816	[2, 1]	1.439692

Table 5: SoyBean

3.1.6 Glass

	Fold	eta	alpha	Error_0_Layers	1_Layers	Error_1_Layers	2_Layers	Error_2_Layers
0	0	0.1	0.0	0.439043	[2]	0.343526	[2, 1]	0.923425
1	0	0.1	0.8	0.146955	[2]	0.490125	[2, 1]	0.962563
2	0	0.1	0.9	1.678813	[8]	0.806928	[8, 2]	0.649085
3	0	0.2	0.0	0.005393	[2]	0.922278	[2, 1]	0.825139
4	0	0.2	0.8	0.087978	[3]	0.707075	[3, 1]	0.786562

Table 6: Glass

3.2 Analysis

	eta	alpha	vector	Error
0	0.1	0.9	[]	2078.525240
1	0.1	0.8	[]	22391.321142
2	0.1	0.9	[]	4913.231924
3	0.1	0.9	[]	1810.918095
4	0.1	0.9	[]	9158.105456
5	0.1	0.9	[]	25794.109549
6	0.1	0.9	[]	23077.420687
7	0.1	0.9	[]	750.874902
8	0.1	0.9	[]	1320.699137
9	0.1	0.9	[]	3351.393042

Table 7: Forest Fires (0 Layers)

	eta	alpha	vector	Error
0	0.1	0.8	[1]	823.704517
1	0.1	0.9	[1]	882.394526
2	0.1	0.8	[1]	921.749307
3	0.1	0.0	[1]	1090.457262
4	0.1	0.0	[1]	1735.753455
5	0.1	0.0	[1]	11014.921992
6	0.1	0.0	[1]	23622.209748
7	0.1	0.9	[1]	356.251943
8	0.1	0.9	[1]	611.648411
9	0.2	0.9	[1]	773.822405

Table 8: Forest Fires (1 Layer)

	eta	alpha	vector	Error
0	0.1	0.0	[1, 1]	819.432223
1	0.1	0.9	[1, 1]	884.218278
2	0.3	0.0	[1, 1]	981.073616
3	0.1	0.9	[1, 1]	1042.042945
4	0.2	0.9	[1, 1]	1624.424382
5	0.2	0.0	[1, 1]	11659.483708
6	0.1	0.0	[1, 1]	23398.755328
7	0.1	0.0	[1, 1]	327.193408
8	0.1	0.0	[1, 1]	606.659521
9	0.2	0.9	[1, 1]	786.348216

Table 9: Forest Fires (2 Layers)

	eta	alpha	vector	Error
0	0.1	0.9	[]	48586.153191
1	0.1	0.9	[]	62051.039789
2	0.1	0.9	[]	95320.213328
3	0.1	0.9	[]	51214.633116
4	0.1	0.9	[]	45355.557689
5	0.1	0.9	[]	39553.515343
6	0.1	0.9	[]	332939.944558
7	0.1	0.9	[]	225346.214707
8	0.1	0.9	[]	69457.146059
9	0.1	0.9	[]	43444.721342

Table 10: Hardware (0 Layers)

	eta	alpha	vector	Error
0	0.2	0.9	[3]	8330.418495
1	0.1	0.0	[2]	17218.662811
2	0.2	0.0	[2]	20812.868837
3	0.1	0.9	[2]	8978.059681
4	0.2	0.0	[1]	7046.141336
5	0.1	0.8	[3]	15403.250371
6	0.3	0.8	[4]	25826.346910
7	0.1	0.9	[1]	49626.412142
8	0.3	0.9	[3]	45011.171885
9	0.3	0.8	[3]	11863.747670

Table 11: Hardware (1 Layer)

	eta	alpha	vector	Error
0	0.2	0.8	[1, 1]	14096.210893
1	0.1	0.8	[2, 2]	12085.431943
2	0.1	0.8	[2, 2]	9785.582632
3	0.1	0.8	[1, 1]	14357.204078
4	0.2	0.0	[1, 1]	22629.769447
5	0.1	0.9	[1, 1]	14375.482539
6	0.1	0.9	[3, 3]	40702.246269
7	0.3	0.8	[3, 3]	79573.696605
8	0.3	0.8	[2, 2]	64984.022955
9	0.2	0.9	[2, 2]	5178.728152

Table 12: Hardware (2 Layers)

	eta	alpha	vector	Error
0	0.3	0.9	[]	3.288676e-03
1	0.2	0.9	[]	3.379399e-08
2	0.3	0.0	[]	2.678647e-03
3	0.2	0.0	[]	1.270868e-03
4	0.2	0.0	[]	1.277039e-03
5	0.3	0.9	[]	1.750001e-03
6	0.3	0.9	[]	8.514205e-04
7	0.3	0.9	[]	2.676298e-04
8	0.2	0.8	[]	5.123353e-03
9	0.3	0.8	[]	6.756407e-03

Table 13: Breast Cancer (0 Layers)

	eta	alpha	vector	Error
0	0.3	0.9	[5]	0.076797
1	0.2	0.0	[2]	0.134487
2	0.1	0.8	[3]	0.723914
3	0.3	0.8	[6]	0.030258
4	0.3	0.0	[3]	0.058774
5	0.3	0.0	[3]	0.090015
6	0.3	0.9	[5]	0.007338
7	0.2	0.8	[4]	0.003744
8	0.1	0.8	[4]	0.074066
9	0.1	0.8	[3]	0.039594

Table 14: Breast Cancer (1 Layer)

	eta	alpha	vector	Error
0	0.3	0.9	[5, 5]	0.103870
1	0.3	0.9	[3, 3]	0.024703
2	0.3	0.9	[3, 3]	0.026115
3	0.2	0.8	[2, 2]	0.534573
4	0.3	0.0	[3, 3]	0.047014
5	0.3	0.9	[3, 3]	0.133094
6	0.2	0.0	[3, 3]	0.078044
7	0.3	0.9	[2, 2]	0.662416
8	0.1	0.0	[4, 4]	0.470755
9	0.2	0.0	[3, 3]	0.327983

Table 15: Breast Cancer (2 Layers)

	eta	alpha	vector	Error
0	0.1	0.0	[]	5.271300
1	0.1	0.8	[]	3.780760
2	0.2	0.8	[]	8.894296
3	0.3	0.8	[]	14.424287
4	0.3	0.8	[]	17.503405
5	0.2	0.0	[]	8.934317
6	0.3	0.0	[]	10.953808
7	0.1	0.8	[]	5.754310
8	0.2	0.8	[]	9.325523
9	0.2	0.8	[]	4.407396

Table 16: Soy Bean (0 Layers)

	eta	alpha	vector	Error
0	0.3	0.9	[1]	1.321726
1	0.1	0.0	[1]	1.348222
2	0.2	0.9	[3]	1.672444
3	0.1	0.0	[3]	1.293673
4	0.1	0.8	[5]	1.775903
5	0.1	0.0	[1]	1.735953
6	0.1	0.9	[4]	2.027785
7	0.3	0.9	[2]	1.493511
8	0.3	0.9	[2]	1.481048
9	0.1	0.8	[1]	1.614733

Table 17: Soy Bean (1 Layer)

	eta	alpha	vector	Error
0	0.3	0.9	[1, 1]	1.431560
1	0.1	0.9	[1, 1]	1.316939
2	0.1	0.8	[2, 2]	1.502644
3	0.2	0.0	[4, 4]	1.429054
4	0.1	0.8	[5, 5]	1.479429
5	0.1	0.9	[1, 1]	1.364127
6	0.1	0.0	[2, 2]	1.754006
7	0.1	0.9	[1, 1]	1.258869
8	0.3	0.9	[2, 2]	1.342824
9	0.3	0.9	[1, 1]	1.351232

Table 18: Soy Bean (2 Layers)

	eta	alpha	vector	Error
0	0.3	0.9	[]	4.649094e-02
1	0.3	0.0	[]	5.692036e-03
2	0.3	0.0	[]	3.243787e-02
3	0.3	0.9	[]	3.340274e-02
4	0.3	0.0	[]	3.515673e-05
5	0.2	0.8	[]	1.096393e+00
6	0.2	0.9	[]	5.659057e-02
7	0.3	0.9	[]	3.571521e-02
8	0.2	0.9	[]	3.740146e-01
9	0.2	0.8	[]	4.206999e-07

Table 19: Glass (0 Layers)

	eta	alpha	vector	Error
0	0.2	0.9	[3]	1.640943
1	0.3	0.9	[2]	1.645674
2	0.3	0.9	[2]	1.609834
3	0.3	0.0	[3]	0.991089
4	0.2	0.9	[3]	1.526899
5	0.2	0.0	[3]	0.898869
6	0.3	0.8	[2]	2.355753
7	0.2	0.9	[5]	1.893859
8	0.3	0.8	[5]	0.640101
9	0.2	0.0	[2]	1.204526

Table 20: Glass (1 Layer)

	eta	alpha	vector	Error
0	0.2	0.9	[3, 3]	1.255075
1	0.1	0.9	[2, 2]	1.590635
2	0.3	0.0	[3, 3]	0.948647
3	0.2	0.9	[3, 3]	0.960102
4	0.3	0.9	[4, 4]	0.725687
5	0.3	0.8	[2, 2]	1.578626
6	0.2	0.9	[5, 5]	1.107097
7	0.3	0.0	[2, 2]	1.096361
8	0.3	0.9	[6, 6]	0.701764
9	0.3	0.0	[3, 3]	0.750968

Table 21: Glass (2 Layers)

	eta	alpha	vector	Error
0	0.1	0.9	[]	178.978643
1	0.1	0.8	[]	219.356454
2	0.1	0.8	[]	233.493604
3	0.1	0.8	[]	171.462608
4	0.1	0.8	[]	314.298906
5	0.1	0.9	[]	211.519099
6	0.1	0.9	[]	213.911654
7	0.1	0.8	[]	171.090093
8	0.1	0.9	[]	159.243591
9	0.1	0.9	[]	196.744165

Table 22: Abalone (0 Layers)

	eta	alpha	vector	Error
0	0.1	0.9	[4]	7.784614
1	0.1	0.0	[5]	6.268733
2	0.1	0.9	[5]	6.246128
3	0.1	0.8	[5]	6.396941
4	0.1	0.9	[5]	5.992276
5	0.1	0.9	[4]	7.042279
6	0.1	0.9	[4]	7.441141
7	0.1	0.9	[7]	6.401624
8	0.1	0.9	[4]	6.171406
9	0.1	0.8	[4]	7.598430

Table 23: Abalone (1 Layer)

	eta	alpha	vector	Error
0	0.1	0.9	[4, 4]	7.172913
1	0.1	0.8	[4, 4]	7.614535
2	0.1	0.0	[4, 4]	8.472752
3	0.1	0.9	[5, 5]	9.460949
4	0.1	0.8	[7, 7]	8.178694
5	0.1	0.8	[3, 3]	8.344943
6	0.1	0.8	[6, 6]	8.236543
7	0.1	0.8	[3, 3]	7.585093
8	0.1	0.0	[5, 5]	9.552205
9	0.1	0.9	[5, 5]	7.531252

Table 24: Abalone (2 Layers)

3.3 Conclusions

3.3.1 Abalone

It seemed that the best learning rate for Abalone was $\eta = 0.1$. Furthermore, momentum seemed to improve performance with an even split between $\alpha = 0.8$ and $\alpha = 0.9$. Performance improved significantly after adding hidden layers to the data. For one layer, the optimal number of nodes was 4 or 5. For layers, the optimal pairs of number of nodes varied more, but it seemed that if the first layer had z nodes, then it was best for the second layer to have z nodes as well.

3.3.2 Forest Fires

Like Abalone, the best learning rate was $\eta = 0.1$. Momentum seemed to usually improve performance, but not as often compared to the Abalone data set. For the times momentum was best to use, $\alpha = 0.9$ was more often better than $\alpha = 0.8$. Adding hidden layers improved performance, and it seemed best to use only one node per hidden layer.

3.3.3 Hardware

Unlike Abalone and Forest Fires, the optimal learning rate varied between the three optional values. Momentum seems to improve performance with an even split between $\alpha = 0.8$ and $\alpha = 0.9$. Adding hidden layers seemed to improve performance, but not to the same degree as in Abalone or Forest Fires data sets. With two layers, it seemed that it was best to have the same number of nodes in each layer.

3.3.4 Breast Cancer

Unlike the previous data sets, a learning rate of $\eta = 0.3$ seems optimal. Overall, momentum seems to improve performance, but there are times momentum was not the best. Unlike the previous data sets, adding hidden layers decreased performance and increased error. Once again, with two layers, having the same number of nodes seemed to be best.

3.3.5 SoyBeans

There seems to be no clear winner for best learning rate among the three choices. Once again, momentum seemed to improve performance overall, with an even split between the two nonzero momentum choices. There was an overall increase in performance when adding hidden layers. With layers, once again, it seemed best to have the same number of nodes in each layer.

3.3.6 Glass

The learning rate seemed best with $\eta = 0.3$, with a close second from $\eta = 0.2$. Overall, momentum seemed to improve performance, with an overall optimal choice of $\alpha = 0.9$. Like in the Breast Cancer data set, the performance seemed to decrease after adding hidden layers. With two layers, it seemed best to have the same number of nodes in each layer.

4. Summary

The most visible increase in performance came from adding momentum to the gradient descent. It is not really clear which of the three learning rates was best. Overall, except for the Breast Cancer and the Glass data sets, adding hidden layers increased performance, but it was not clear that one hidden layer was better or worse than two hidden layers. The number of nodes per layer that seemed best seemed to vary between 1 to 5, though 6 or 7 sometimes was best, too. If there were two layers, it seemed best to have the same number of nodes per layer.

We were surprised that varying the learning rates among $\{0.1, 0.2, 0.3\}$ did not noticeably affect the errors. There were many times that 0.3 was optimal, contrary to our hypothesis. As we predicted, momentum seemed to overall increase performance. Furthermore, adding hidden layers seemed to increase performance, as we predicted, but were surprised that it did not for Breast Cancer and Glass data sets. We were also surprised that two layers did not noticeably–if at all–beat the performance of having only one hidden layer.