Back propagation Project

CS478 – Machine Learning
Allen Kim

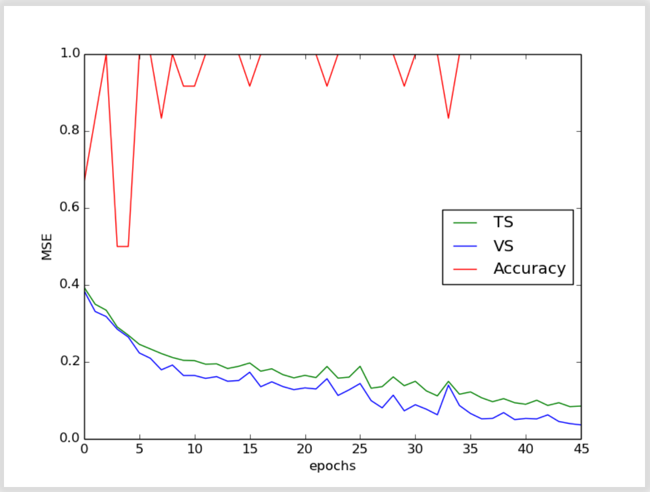
1. Implement the back propagation algorithm and integrate it with the toolkit.

Training: For this lab I used JAVA and the provided tool kit for JAVA. First I start with initializing the weights of nodes in the layers. To initialize the weights for each nodes, I used random.nextDouble() and multiply by 2*range and minus range, which place the random weights between plus and minus of the range. I created the global variable for the number of hidden layers, the number of nodes in each hidden layers, and number of nodes for output layer to have the ability to create an arbitrary structure. After initialize all the weights, I split the training features and training labels into training set and validation set. I made the training set to shuffle after every epoch. For actual back propagation part, I used the recursion function to run forward until it reaches the output layer to calculate the predicted output. Then, as the recursion function recurs back it calculates the updated weight values. When the function ends, I called the update function to update all the weights for each node. Also if the momentum option is on, I updated the weights accordingly.

Testing: For the testing, I also ran the recursive function to reach all the way to the output layer. When it reaches the output layer, I saved the highest output index and compared with the actual output index.

2. Use your back propagation learner, with stochastic weight updates, for the iris classification problem.

For this part, I used one hidden layer with 8 nodes, which is the twice number of inputs. I separated the whole features with 75 percent of training set and 25 percent of testing set. The learning rate is 0.1.



Stopping Criterion: I kept track of the best solution so far (bssf). If the bssf of the validation set are not just improving but also not changing for 10 or more times with less than 0.001 differences, it stops. The graph above shows that it stopped at the accuracy of 1 with 45 epochs, because the accuracy stayed at 1 for more than 10 times. The test set accuracy was 94 percent.

Note: This graph seems reasonable to me, because the MSE for training set and validation set are decreasing every epoch, which tells that we are learning correctly and weights are adjusted right. My concern was on the accuracy, because it is jumping very much. My guess on why it is happening is that because the training set is not enough and we don't use the validation set as the training data, the weights do not fit perfectly to validation data. As the weight updated by running back propagation algorithm, it is trying to fit more for training data set. However, after enough epochs the accuracy of validation set settles and in our case it reached to all the way to 100 percent.

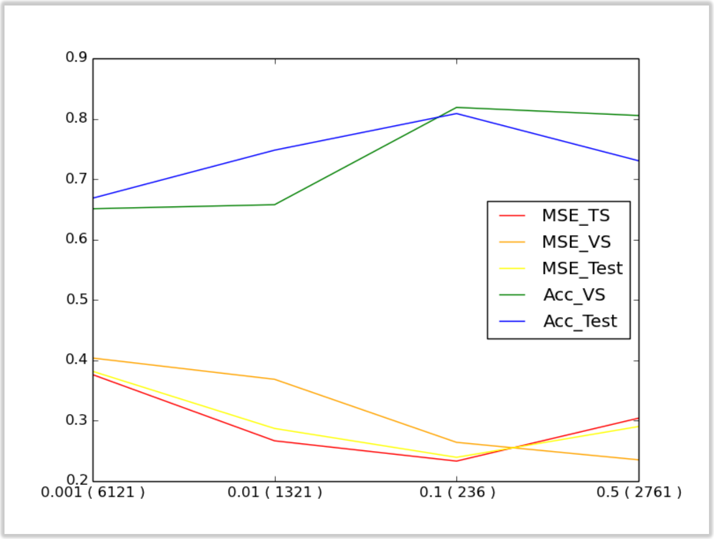
3. For 3-5 you will use the vowel data set, which is a more difficult task

- Consider carefully, which of the given input features you should actually use.

	1st	2nd	3rd	4th	5th	6th	SUM
Train/Test	0.04	-0.12	-0.25	0.07	-0.08	-0.58	-0.91
Speaker	0.09	0.16	0.04	-0.03	-0.02	0.08	0.31
Gender	-0.01	-0.05	0.14	0.02	-0.10	0.37	0.37

Note: I the table above contain the average weight from each input node to the first hidden layer. I tried them 6 times and sum the averages. As we can see, the feature, "Train/Test", has the lowest sum. Since the bigger the weight is, the more affects to the next layer, I can conclude that the first feature has the least effect to the output, and second and third features are quite effective to the result.

- Create one graph with MSE for the TS, VS, and test set, the classification accuracy on the VS and test set, and the number of epochs needed to get to the best VS solution, on the y-axis (6 total items with 3 different scales!), with the different learning rates on the x-axis.

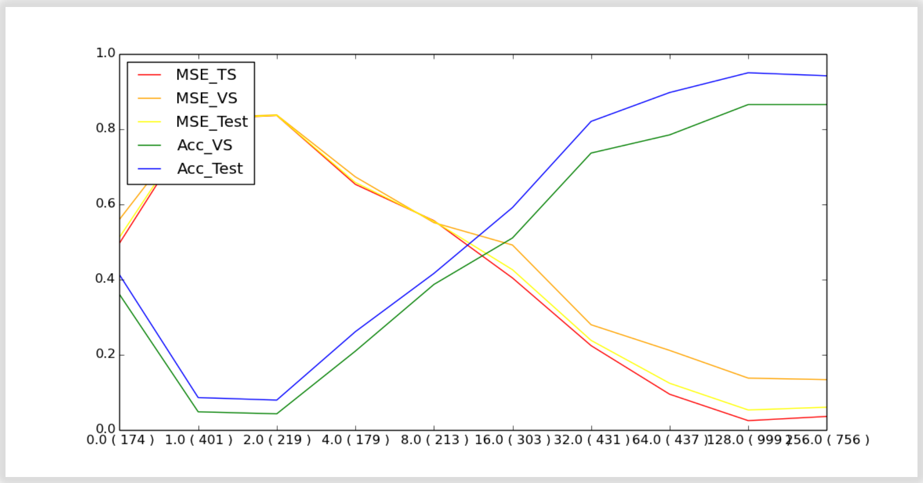


Stopping criterion: I have changed my stopping criterion this time. This time, I stored all 100 of previous accuracy and calculated the minimum. And, I compare the minimum with the current accuracy. If the current accuracy less than minimum of previous 100 accuracies, I stopped. It works because if training stopped, that means there was no improvement and actually decreased than last 100 accuracies.

Note: I have tried with 4 different learning rates: 0.001, 0.01, 0.1, and 0.5. The graph itself is pretty much self-explanatory, but I think you might be confused with the x-axis. X-axis has the number and another number inside of parentheses. First numbers are the learning rate, and the number inside of parentheses are the epochs, which was required to stop. The stopping criterion is the same as the last example. I made it to stop at the point where the validation set's accuracy settles and not make much progress. By doing this experiment I could see the effects on the different learning rates. My first guess was that lesser learning rate would help the machine to reach higher accuracy and lesser MSE. The result was little different than my expectation. Actually the accuracy was the best with learning rate of 0.1 and MSE was the lowest. This concludes that learning rate of 0.1 give me the best result. Since 0.5 gives the higher MSE, I stopped experimenting at the learning rate of 0.5. Also the learning rate of 0.1 requires the least epochs.

4. Using the best LR you discovered, experiment with different numbers of hidden nodes.

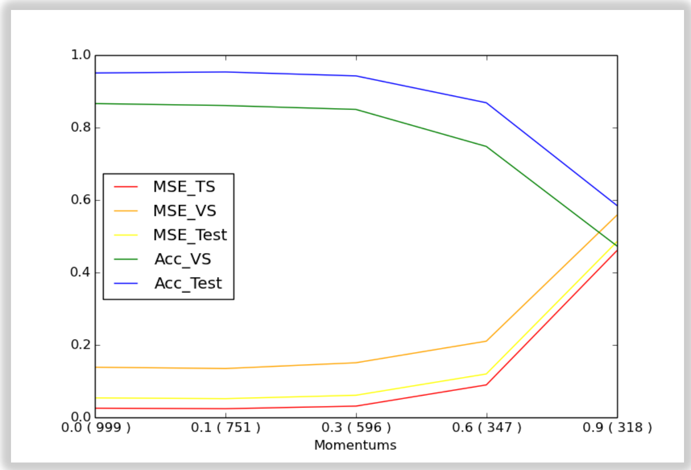
- Start with 0 hidden nodes, then 1, and then double them for each test until you get no more improvement.



<u>Stopping criterion</u>: The same as the last one, since I am using the same algorithm.

<u>Note</u>: I started with 0 nodes, which made no hidden layer. The validation set accuracy for no hidden layer was quite low, but better than 1 node case. I think it works as perceptron instead of back propagation. With 1 or 2 nodes gave me the lowest accuracy. As we added more number of nodes, it increased accuracy excellently. The increase stopped with 128 nodes and there were no improvement later. In conclusion, I think 128-nodes gives enough accuracy we need with 1 hidden layer. Also as we use more number of nodes, each epoch takes longer time, so I stopped at the reasonable accuracy and less number of nodes possible.

5. Try some different momentum terms in the learning equation using the best number of hidden nodes and LR from your earlier experiments.



Stopping criterion: The same as the last one, since I am using the same algorithm.

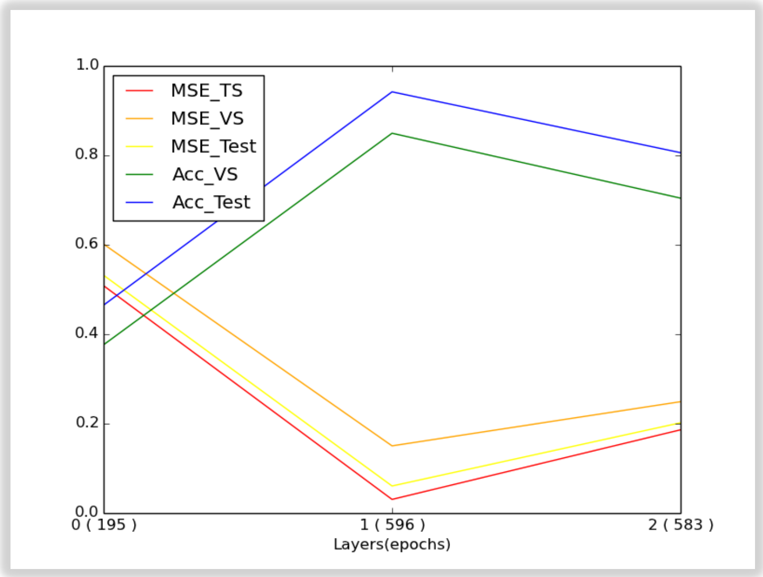
Note: This graph explains the effects of having momentum in the training. We can see that after momentum 0.3 the accuracy decreases much and MSE increased much, so I could tell the momentum greater than 0.3 is not very helpful; however, the momentum between 0 and 0.3 has very similar accuracies. In this case, I had to decide what momentum to choose between them. I think momentum 0.3 works the best in our case, because it has the least epochs, which means it took the least time. If we can get the great accuracy with the less time spent, I think that is the best solution. In conclusion, the momentum of 0.3 is the best in our case.

6. Do an experiment of your own regarding back propagation learning. Do something more interesting than just trying BP on a different task, or just a variation of the other requirements above.

At this point I wanted to know that with our optimized learning algorithm what would be the difference in time, epoch, and accuracy (test set) depends on the number of layers. So I was going to have from 0 layers to go more and more layers until I reach a point to stop improving. There is my table of the result below:

# Layers	Time	Epochs	Accuracy
0	1.1 seconds	195	40%
1	20 seconds	596	90%
2	88 seconds	583	72%

As we can see after one layer, everything went down and took so much more time. I thought that for 2 or 3 layers would give us better result, but it was not right. It was also very interesting to see the accuracy actually went down with more layers. To see the better insights on this experiment, I created a graph same as the previous exercises.



The graph was quite straightforward. As I check from the table above, the accuracy and MSE are not improving at all after one layer. Since one machine learning algorithm cannot work for all different types of data. For this data, I will conclude that it works best with one layer, the learning rate of 0.1, the nodes' number of 128, and the momentum of 0.3.

For this task, I was able to make a lot better accuracy than perceptron algorithm. I can tell that even though back propagation algorithm is not perfect for every data type, but it can be applied to more complicate data sets.