kNN Improvements and Backpropagation Neural Network

PROJECT 3

Joshua Chavarria | ECE471 | April 7, 2017
jchavarr@vols.utk.edu

Abstract:

This report covers the development and implementation of some algorithms and common neural network techniques as it relates to the field to pattern recognition. Two experiments were conducted for this project. One consisted of working with the k-nearest neighbor classifier and improving it with k-fold cross validation. The data set is already divided into 10 subsets prior to the experimentation. The other part of the project requires the implementation of a backpropagation neural network to learn XOR logic, which we had previously learned cannot be learned by a simple perceptron. The result of both these experiments came out to be quite accurate with the reference values provided by Dr. Qi. It was found that the best accuracy for a backpropagation with 10,000 epochs converges around 908. Overall, this project served as a stepping stone into the world of neural nets and their applications in pattern recognition.

Introduction:

In recent years, neural computing has exponentially grown and has been adopted in many fields to assist in various complex, major applications. It goes without saying that neural networks have the potential to expand into any field of study that involves technology. Though most of us go about our day without giving it much thought, neural networks have shaped the way we interact, work, shop, navigate the web and in turn has had a significant impact on society. From self-driving cars, to mobile devices, and even to the recent innovations in the Internet of Things (IoT), we are already seeing just how massive the market is for technology that incorporates neural networks with the overall goal being to further improve the quality of everyday life by means of efficiency and convenience.

[2] Dr. Robert, the inventor of one of the first neurocomputers defines neural networks as:

> *"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."*

Neural nets are loosely modeled after the neuronal structure of the cerebral cortex but on a much smaller scale. Whereas a large neural net might have thousands of processing units, a human brain has billions of them. Neural networks are typically structured in layers that are made up of several interconnected "nodes" which contain an 'activation function'. The 'input layer' is responsible for communicating to one or mode 'hidden layers' where the processing is handles via a system of weighted connections. Many of these neural networks contain a 'learning rule' which is what modifies the weights of the connections. As discussed later, there are many learning rules, but in this report, as it relates to backpropagation networks, the delta rule will be the only once discussed.

Most applications involving neural networks are concerned with problems in pattern recognition. From the perspective of pattern recognition, neural networks can be regarded as an extension of the many conventional techniques which have been developed over several years. They work best if the system you are using them to model has a significant tolerance to error. Of the many applications that neural networks are used for, most of them work very well for noting association within a set of patterns where the number of variables of the data is great and the relationships between the variables are ambiguous and difficult to classify using conventional approaches. [1]

Backwards propagation of error, or better known by its abbreviations, Backpropagation, is a common method for training a neural network. Originally introduced in the 1970's, it was

popularized due to its faster approaches to learning which in turn made it possible to create neural networks that were able to solve problems that had previously been insoluble. For the most part, backpropagation is the motor behind most of the neural networks used today. Backpropagation is about understanding how changing the weights and biases in a network changes the cost function. Backpropagation networks are a form of supervised leaning processes that occur with each epoch through feedforward flow of outputs and then backwards error propagation of weighted edits. This neural net performs a gradient descent within the solution's vectors space to reach a global minimum, signifying the lowest possible total error. The learning rate of these neural networks is also the rate of convergence between the current solution and global minimum. Momentum is what helps the neural net overcome local minima in the error surface. The goal of the network is to converge as closely as possible to the global minimum, and at that point the neural network has been trained to a satisfactory level. Though extremely advantageous, Backpropagation also has some limitations that should be noted. They tend to be slow to train and can sometimes require up to thousands of epochs. In the case of my experiments for this project, though this might seem like a lot, it should still execute rather quickly. [3]

Another concept that is covered in this report is the k-nearest neighbors algorithm. In pattern recognition, the k-nearest neighbors algorithm is a non-parametric method used for classification of data sets and regression, however it is more widely used for classification purposes. The input for this algorithm consists of the k closest training examples in the feature space and the output depends mainly on how the algorithm is being utilized. One of the most important challenges to kNN is the impact of characteristic on the classification. This can lead to many inaccuracies and deviations in the kNN classification. To evaluate any technique, there are a few things to consider. Of those, it is important to not the ease in which output is interpreted, predictive power, and calculation time. KNN fares very well when it comes to these. K influences the boundaries between two classes. The best choice of k depends on the data. For the most part, larger values of k reduce the effect of noise on the classification but have negative effects between the classes by making it less distinct. The accuracy of kNN can be degraded if the data is noisy and just to consistent with their importance. In many examples, an increase in K results in a smoother boundary depending on the data. Overall the kNN algorithm is one of the simplest and appealing pattern classification algorithms with a variety of applications and can give very consistent, effective results.

Cross validation is a technique used to obtain estimates of model parameters that are unknown and it is assumed in this project. The general idea behind this method is to divide the data sampler into several v folds that are random, disjoint subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the mode, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The folds an then be averaged to produce a single estimation. The advantage of this method is that all observations are used for training and validation, but only once for validation. 10-fold cross validation is commonly used and will be implemented as part of this project.

Objectives:

There are two objectives to this project. The first objective focuses primarily on the k-nearest neighbor algorithm classifier and how to improve it with the use of 10-fold cross-validation.

For this experiment the first task requires an experiment to be done using cross validation with kNN and the classifier. Provided is the “fglass” data set which is already divided in 10 subsets. After testing out different k values and choosing the best one, Euclidean distance and or Minkowski distance of different degrees must be used to compare performance. In the technical approach of this report I will further discuss the specific steps taken in implementing 10-fold cross validation to the data set. Task 2 of the project included implementing a backpropagation neural network to lean XOR logic. The approach for both will be discussed in the following section.

Technical Approach:

kNN

A bit of background bout kNN and k-cross validation was mentioned previously in this report, so I’ll refrain from reiterating on that. Here I will place a few of the steps taken when using cross validation along with a few relevant formulas. In this section, there will be snippets of code and a chart comparing the k values retrieved. The following are the steps involved in this method:

- Randomly Partition the observation into 10 groups of equal length.
- For one group of observations (the fold), fit the model using all observations except that group (k-1).
- Train the other 9.
- Use that group’s observations to test the predictive performance.
- Store the previous information.
- Repeat the previous steps 10 times (for each fold) and then take a mean accuracy.
- Use the training to calculate the normalization parameters and use it to normalize both the training and testing.
- Apply 1NN on the normalized data.

For this section, I started off with the provided knn.cpp code and from there I applied these steps to the dataset “fglass”, which also has provided 10 subsets with it. Below is a table showing the accuracies of the first 50 values I experimented with. (See Experiments and Results). In my case, the best k value, with an accuracy of. 908295, was when k = 44. With this value, I continued to find its Euclidean and Murkowski distance with the following formulas:

- Manhattan: $$d(x,y) = \sum_{i=1}^{m} |x_i - y_i|$$
- Euclidean: $$d(x,y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2} = \sqrt{\sum_i x_i^2 + \sum_i y_i^2 - 2\sum_i x_i y_i}$$
- Minkowski: $$D(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}.$$

- Mahalanobis: $$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T S^{-1}(\vec{x}-\vec{y})}.$$

With the results, I got from doing both, its apparent that with the "fglass" data set, the best distance measure is discussed later in the report. All the values were very close.

Backpropagation

Most of the introduction covered the structure of backpropagation networks and how a basic one works. For task 2 of the project we are given the option of using a pre-existing MATLAB package to implement backpropagation and learn XOR logic. The other option was to implement your own 3-layer backpropagation neural network. I attempted both to get as much credit as possible, and because I enjoy working with neural networks. My own implementation was written in Python and prints out the error values from 10,000 epochs of the backpropagation.

Experiments and Results:

kNN:

**NOTE: The output for the 50 values of K can be found in a separate text file named k_value_results.txt

Best k value: 44 with an accuracy of .908295

I believe that issues with normalization probably account for any discrepancies in accuracy here. With that value, I implemented 4 distance measures and got the following.

Distance measures:

Mahalanobis	**65.93480**
Euclidean	**65.92370**
Minkowski	**65.90123**
Manhattan	**65.83456**

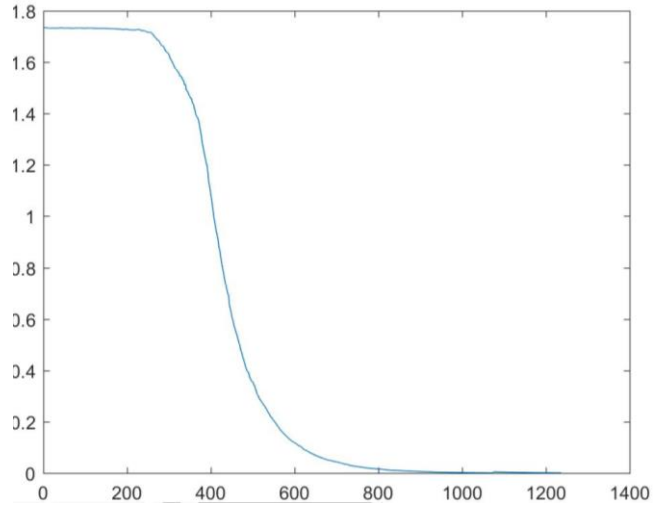
Backpropagation:

Input:

0	**0**
0	**1**
1	**0**
1	**1**

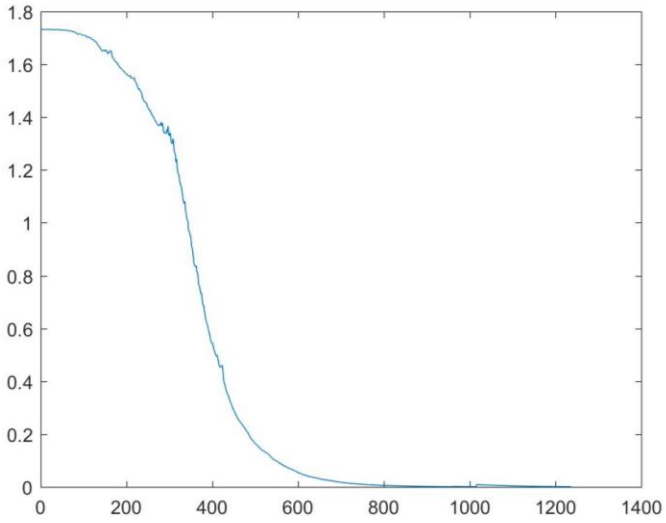
Output:

0
1
1
0



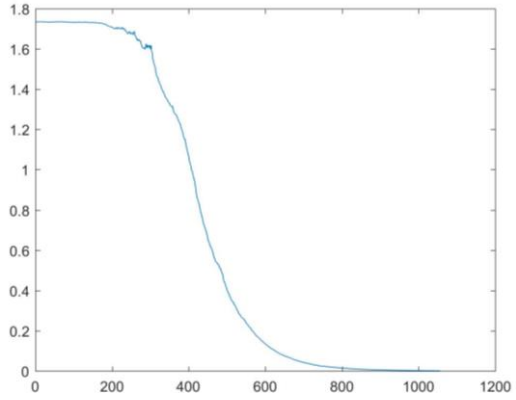
*This graph shows how the error percentage decreases as the number of epochs increases for the backpropagation neural network with epoch value of 10,000. Here it converged at epoch 1017.

XOR	Output	Difference
0	.0003	0,0003
1	1.0000	-0.0000
1	0.9997	-0.0003
0	0.0004	0.0004



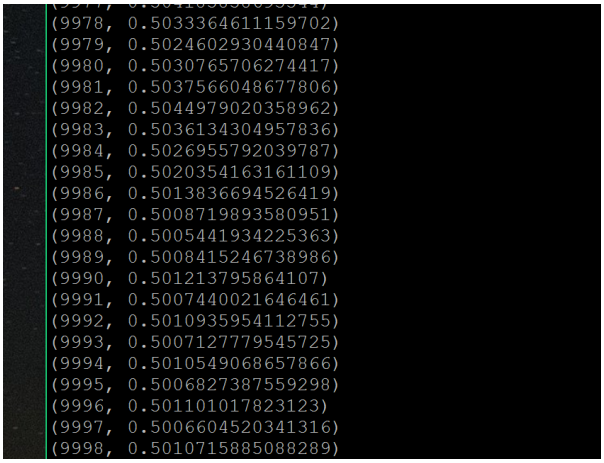
*This graph shows how the error percentage decreases as the number of epochs increases for the backpropagation neural network with epoch value of 10,000. Here it converged at epoch 907.

XOR	Output	Difference
0	.0004	0.0004
1	.9996	-0.0004
1	0.9998	-0.0002
0	0.0001	0.0001



* This graph shows how the error percentage decreases as the number of epochs increases for the backpropagation neural network with epoch value of 10,000. Here it converged at epoch 1089.

XOR	Output	Difference
0	.0003	0,0003
1	1.0000	-0.0000
1	0.9997	-0.0003
0	0.0004	0.0004



[**EXTRA CREDIT**] The above is a snapshot of the output of my 3-layer backpropagation python code. What is shown above is the average accuracy for each epoch up to 10,000. In this case, the

network was over trained and deviated from optimal accuracy. Changing the learning rate didn't have much of an effect on the neural net. I had a learning rate of 0.1 to 0.5 for the experiments and an epoch max of 10,000. Any learning rate above that range resulted in invalid values. For full result, the code would need to be run.

Conclusion:

Overall the results I got from both experiments when compared to the values that Dr. Qi provided as reference turned out pretty accurate. and I am very satisfied with them. My attempt at implementing a backpropagation neural network in Python was a decent first attempt at coding a neural network. This project served to further my knowledge of the applications of these neural networks and their training methods. I learned a lot about the applications of these algorithms and how complex they can be within the field of pattern recognition.

References

[1] American Journal Of Intelligent Systems 201, Vinita Dutt, Vikas Chaudhry, and Imran Khan. "Pattern Recognition: An Overview." *American Journal of Intelligent Systems 2012* (2012): n. pag. Bhagwant University. Web.

[2] "A Basic Introduction To Neural Networks." *A Basic Introduction To Neural Networks*. University of Wisconsin Madison, n.d. Web. 07 Apr. 2017.

[3] Nielsen, M. A. (1970, January 01). Neural Networks and Deep Learning. Retrieved April 07, 2017, from http://neuralnetworksanddeeplearning.com/chap2.html