Neural Network Architecture for Incremental Learning

by

Sanchit Alekh- IIT2012108

Agrim Khanna - IIT2012203

Anmol Jagetia - IIT2012163

Abhinav Gupta - IIT2012162

Devansh Sharma - IIT2012204

Vaibhav Choudhary - IIT2012100

under the guidance of

Prof. Sudip Sanyal

Indian Institute of Information Technology

Devghat, Jhalwa

Allahabad - 211012

Uttar Pradesh, India

Contents

Acknowledgements

The authors would like to thank **Prof. Sudip Sanyal** for planting the idea of the project, as well as providing invaluable support and feedback throughout the course of our work. He was very instrumental in guiding us and suggesting the use of the Adaptive Learning Technologies which form a defining part of our project. Working with Prof. Sanyal has been an enlightening experience for all of us, and our heartfelt reverence goes to him for having mentored us in our project.

Date: March 14th 2015

Certificate

This is to certify that **Sanchit Alekh, Abhinav Gupta, Anmol Jagetia, Agrim Khanna, Devansh Sharma and Vaibhav** have successfully worked on the project titled **Neural Network Architecture for Incremental Learning** under the guidance of **Prof. Sudip Sanyal** for their Sixth Semester Mini Project at Indian Institute of Information Technology, Allahabad

This Project Report is the proof and record of the bonafide research work carried out by them from January-May 2015.

Sudip Sanyal

Professor

Indian Institute of Information Technology Allahabad

Date: March 14th 2015

Abstract

Traditional Classification and Learning Models are incapable of assimilating and incorporating novel experiences from the environment that can not be classified with the existing knowledge. The human brain, on the contrary works quite differently. Humans beings have an intentional and attentional learning mechanism, by the virtue of which they are able to not only adapt, but handle new situations efficiently as well. Therefore, our approach tries to model this inherent characteristic of the human brain, which is now studied under **Incremental Learning**. We have tried to compare a Neural Network Architecture that uses **Adaptive Resonance Theory** with an Ensemble approach, which tries to explain the functioning of the human brain in an abstract manner.

Keywords: Incremental Learning, Machine Learning, Adaptive Resonance Theory, Cognitive Science

Chapter 1

Introduction

The objective of this project is to study the different approaches that achieve Incremental Learning. An incremental learning algorithm is required to work incrementally, that is they don't have the access to the history of information presented to them over time. They essentially solve the problem of stability-plasticity dilemma, the problem whereby the brain, or algorithm in this case, learns quickly and stable without catastrophically forgetting it's past knowledge.

In this project two approaches of incremental learning, namely Fuzzy ARTMAP and Learn++, have been considered for a classification problem and the results have been juxtapose with traditional approach of classification and pattern recognition, the Multi-Layer Perceptron.

Fuzzy ARTMAP is a class of neural network architecture that performs incremental learning of recognition categories using adaptive resonance theory, in response to input vectors presented in arbitrary order. The dynamics of a Fuzzy ARTMAP are described in terms of fuzzy set-theoretic operations.

Learn++ is an algorithm that is based on the AdaBoost algorithm. The essence of this algorithm is a technique called boosting, that takes two weak classifiers that can perform marginally better than random guessing for a two class problem and their synthesis transforms into a strong classifier that has low-error rates. When boosting is extended to incorporate multi-class problem and regression problem the resulting algorithm is the Adaboost. By combining an ensemble of classifiers, Learn++ achieves incremental learning, as well as performance improvement over Adaboost.

Chapter 2

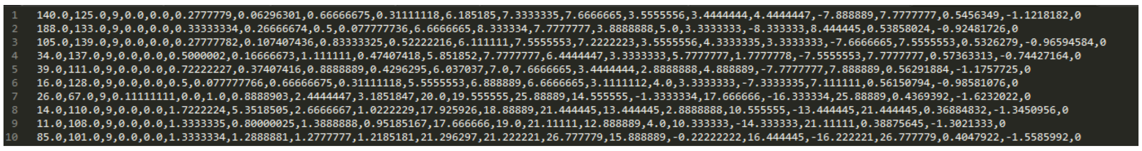
Fuzzy ARTMAP

Fuzzy ARTMAP is an approach based on adaptive resonance theory that performs supervised incremental learning. The initial ARTMAP system was used to classify binary vectors[2]., while a Fuzzy ARTMAP is a more general system that can accommodate binary as well as multi-class data. The impetus of calling it fuzzy is that the dynamics of this system are described in fuzzy set-theoretic operations.

Also introduced is an ARTMAP voting strategy. This voting strategy is based on the observation that ARTMAP fast learning typically leads to different adaptive weights and recognition categories for different orderings of a given training set, even when overall predictive accuracy of all simulations is similar. The voting strategy uses an ARTMAP system that is trained several times on input sets with different orderings. The final prediction for a given test set item is the one made by the largest number of simulations. Since the set of items making erroneous predictions varies from one simulation to the next, voting cancels many of the errors.

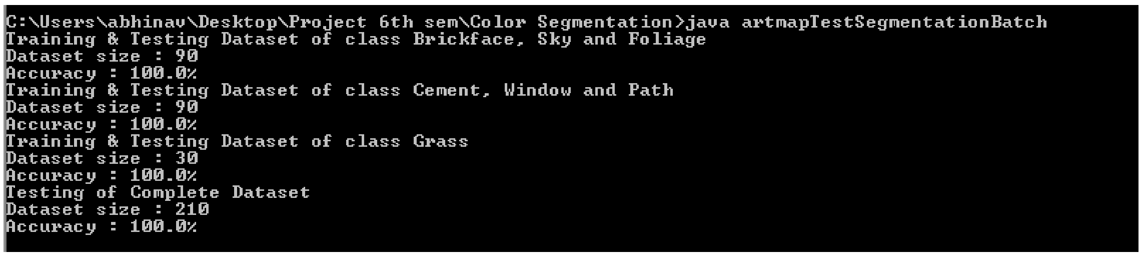
The base fuzzy ARTMAP implementation was taken from the University of Boston's software repository of Cognitive and Neural Systems (CNS) Department[1]. This ARTMAP module implements a range of ARTMAP versions including fuzzy ARTMAP.

A number of datasets were tested on the fuzzy ARTMAP module which included Iris dataset, Poker-hand dataset and the Image segmentation dataset. The module



```
140.0,125.0,9,0.0,0.0,0.2777779,0.06296301,0.66666675,0.31111118,6.185185,7.3333335,7.6666665,3.5555556,3.4444444,4.4444447,-7.888889,7.7777777,0.5456349,-1.1218182,0
188.0,133.0,9,0.0,0.0,0.33333334,0.26666674,0.5,0.077777736,6.6666665,8.333334,7.7777777,3.8888888,5.0,3.3333333,-8.333333,8.444445,0.53858024,-0.92481726,0
105.0,139.0,9,0.0,0.0,0.27777782,0.107407436,0.83333325,0.52222216,6.111111,7.5555553,7.2222223,3.5555556,4.3333335,3.3333333,-7.6666665,7.5555553,0.5326279,-0.96594584,0
34.0,137.0,9,0.0,0.0,0.5000002,0.16666673,1.111111,0.47407418,5.851852,7.7777777,6.4444447,3.3333333,5.7777777,1.7777778,-7.5555553,7.7777777,0.57363313,-0.74427164,0
39.0,111.0,9,0.0,0.0,0.72222227,0.37407416,0.8888889,0.4296295,6.037037,7.0,7.6666665,3.4444444,2.8888888,4.888889,-7.7777777,7.888889,0.56291884,-1.1757725,0
16.0,128.0,9,0.0,0.0,0.5,0.077777766,0.66666675,0.31111118,5.5555553,6.888889,6.6666665,3.1111112,4.0,3.3333333,-7.3333335,7.111111,0.56150794,-0.98581076,0
26.0,67.0,9,0.11111111,0.0,1.0,0.8888903,2.4444447,3.1851847,20.0,19.555555,25.88889,14.555555,-1.3333334,17.666666,-16.333334,25.88889,0.4369392,-1.6232022,0
14.0,110.0,9,0.0,0.0,1.7222224,5.3518505,2.6666667,1.0222229,17.925926,18.88889,21.444445,13.444445,2.8888888,10.555555,-13.444445,21.444445,0.36884832,-1.3450956,0
11.0,108.0,9,0.0,0.0,1.3333335,0.80000025,1.3888888,0.95185167,17.666666,19.0,21.11111,12.888889,4.0,10.333333,-14.333333,21.11111,0.38875645,-1.3021333,0
85.0,101.0,9,0.0,0.0,1.3333334,1.2888881,1.2777777,1.2185181,21.296297,21.222221,26.777779,15.888889,-0.22222222,16.444445,-16.222221,26.777779,0.4047922,-1.5585992,0
```

Figure 2.1: Image segmentation Training data set.



```
C:\Users\abhinav\Desktop\Project 6th sem\Color Segmentation>java artmapTestSegmentationBatch
Training & Testing Dataset of class Brickface, Sky and Foliage
Dataset size : 90
Accuracy : 100.0%
Training & Testing Dataset of class Cement, Window and Path
Dataset size : 90
Accuracy : 100.0%
Training & Testing Dataset of class Grass
Dataset size : 30
Accuracy : 100.0%
Testing of Complete Dataset
Dataset size : 210
Accuracy : 100.0%
```

Figure 2.2: Output of Fuzzy ARTMAP for batch training of Image Segmentation

is sensitive to factors including noisy data, type of attributes i.e. categorical or real data and the size of the training dataset.

Chapter 3

LEARN++

Learn++ inspired in part by the ensemble structure of the AdaBoost.M1 algorithm, exploits the synergistic expressive power of an ensemble of classifiers to incrementally learn additional information from new data. Specifically, for each database that becomes available, Learn++ generates a number of diverse classifiers, which are then combined using a suitable combination rule (originally, the weighted majority voting) [3].

For each iteration t during current dataset D_k, training (TR) and testing (TE) subsets are randomly generated from the current dataset.These subsets are provided to WeakLearn (in our case, CART algorithm), which returns the hypothesis h_t. Unlike AdaBoost.Ml, the error, e, is computed from the misclassified patterns of TR + TE. If $E > 1/2$, h_t is discarded, and new TR and TE are generated. The weighted majority voting is then called to compute the overall hypothesis, H_t, of all hypotheses generated. Also unlike AdaBoost.Ml, the overall hypothesis, H_t, and its scaled error, B_t(beta) are then used to update the distribution. Note that the weight of voting for each h, is still based on its own scaled error B_t [4].

3.1 How is LEARN++ incremental ?

Learn++ creates a composite hypothesis Ht representing the ensemble decision, and uses the ensemble performance to update its weight distribution.

When instances of a new class are introduced, an existing ensemble Ht is bound to misclassify them, forcing the algorithm to focus on these instances that carry novel information.

For a weight update rule based on the performance of ht only, the training performance of the first ht on instances from the new class is independent of the previously generated classifiers. Therefore, the new ht is not any more likely to misclassify new class instances[3].

Chapter 4

Multi-Layer Perceptron Neural Networks

The backpropagation algorithm looks for the minimum of the error function in weight space using the method of gradient descent. The combination of weights which minimizes the error function is considered to be a solution of the learning problem. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the conti- nuity and differentiability of the error function. Obviously we have to use a kind of activation function other than the step function used in perceptrons, because the composite function produced by interconnected perceptrons is discontinuous, and therefore the error function too. One of the more popular activation functions for backpropagation networks is the sigmoid, a real function sc : $IR(0, 1)$ defined by the expression

$$s_c(x) = \frac{1}{1 + e^{-cx}} \tag{4.1}$$

The constant c can be selected arbitrarily and its reciprocal 1/c is called the temperature parameter in stochastic neural networks. The shape of the sigmoid changes according to the value of c.

The graph shows the shape of the sigmoid for c = 1, c = 2 and c = 3. Higher values of c bring the shape of the sigmoid closer to that of the step function and in the limit c the sigmoid converges to a step function at the origin. In order to simplify all expressions derived in this chapter we set c = 1, but after going through this material the reader should be able to generalize all the expressions for a variable c. In the following we call the sigmoid s1(x) just s(x).

Many other kinds of activation functions have been proposed and the back- propagation algorithm is applicable to all of them. A differentiable activation function makes the function computed by a neural network differentiable (assuming that the integration function at each node is just the sum of the inputs), since the network itself computes only function compositions. The error function also becomes differentiable. Since we want to follow the gradient direction to find the minimum of this function, it is important that no regions exist in which the error function is completely flat. As the sigmoid always has a positive derivative, the slope of the error function provides a greater or lesser descent direction which can be followed. We can think of our search algorithm as a physical process in which a small sphere is allowed to roll on the surface of the error function until it reaches the bottom.

4.0.1 Back-propagation Algorithm

Consider a network with a single real input x and network function F. The derivative F(x) is computed in two phases:

Feed-forward: The input x is fed into the network. The primitive func- tions at the nodes and their derivatives are evaluated at each node. The derivatives are stored.

Backpropagation: The constant 1 is fed into the output unit and the network is run backwards. Incoming information to a node is added and the result is multiplied by the value stored in the left part of the unit. The result is transmitted to the left of the unit. The result collected at the input unit is the derivative of the network function with respect to x.

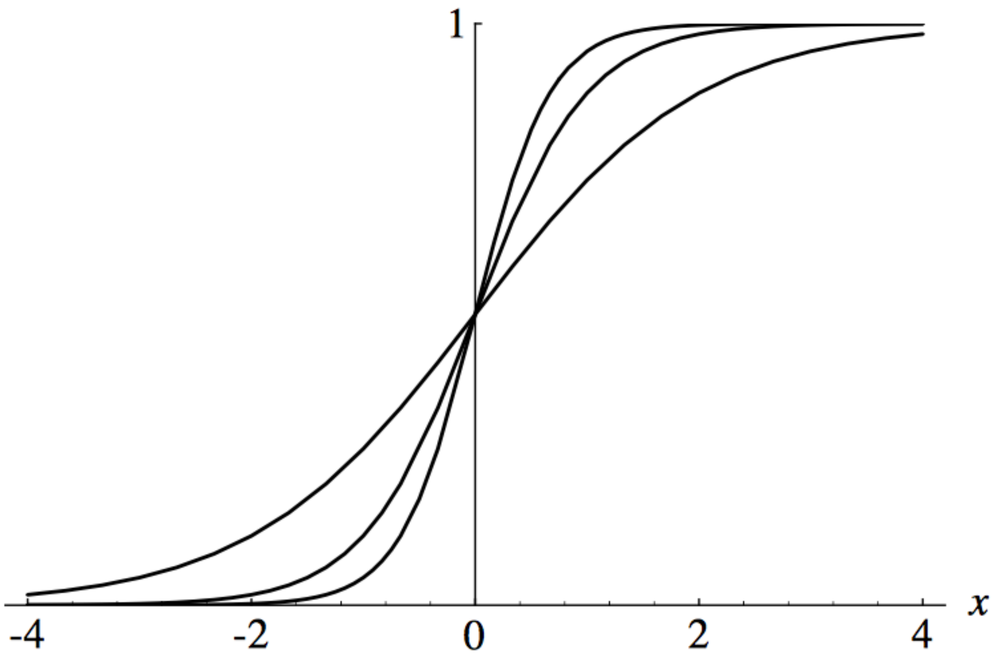


Figure 4.1: A sigmoidal transfer function used in back propagation.

Chapter 5

Result

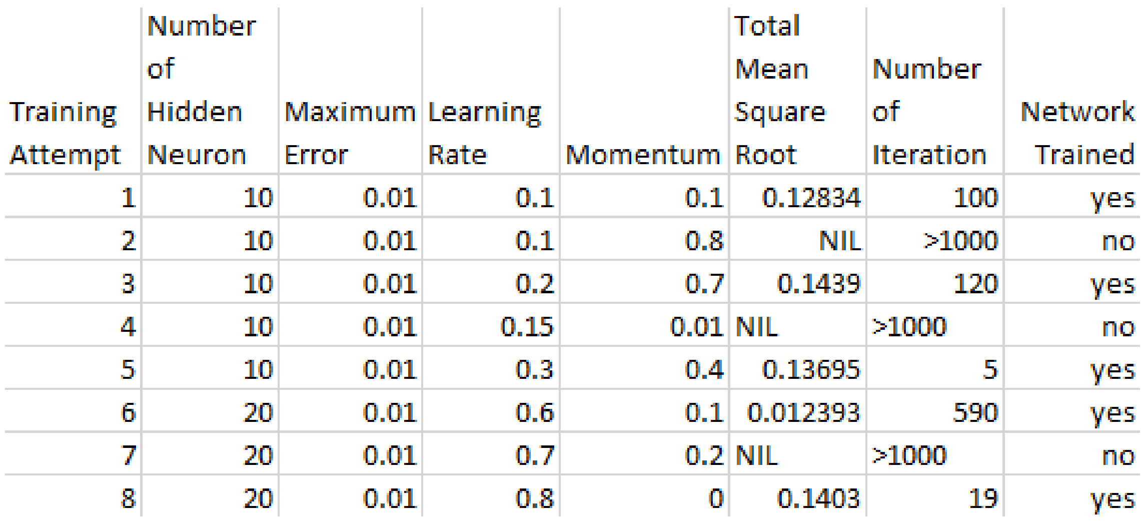
Our project aims at comparing the three approaches towards Pattern Classification, two of which, namely, ARTMAP and LEARN++ allow for incremental learning of data. We used a total of four standard datasets provided by University of California, Irvine for comparing these approaches.

In our choice of datasets, we have tried to test our algorithms on the most stringent set of parameters possible. Poker Hands dataset is characterized by its unusually large testing samples, which are 1 million. Therefore, testing our algorithm on Poker Hands gives us an extensive feedback about the robustness of the algorithm. Barring the Ionosphere dataset, all the other datasets are for multi-class classification.

5.0.2 Error Backpropagation Neural Networks

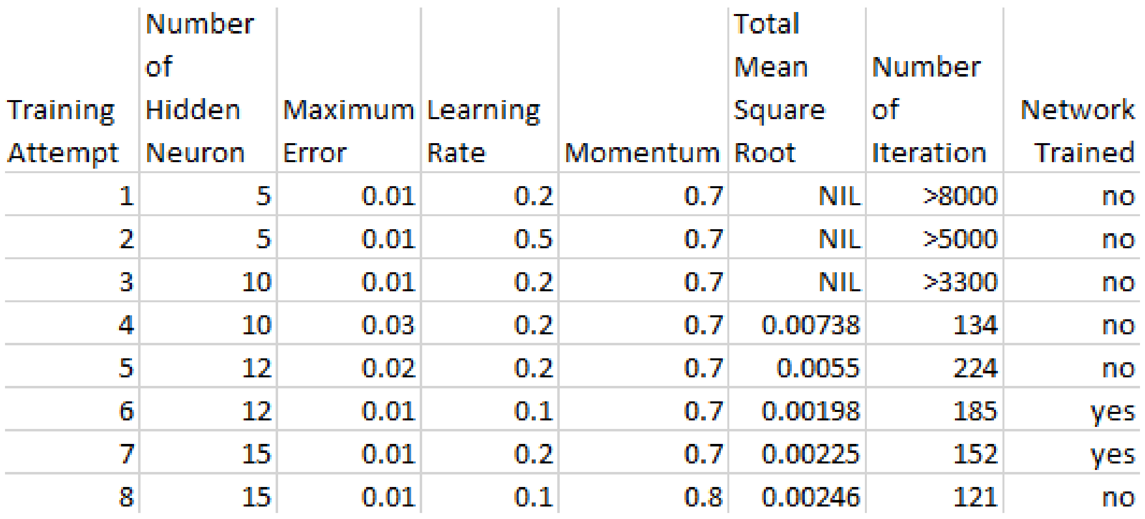
We tested the Backpropagation neural networks extensively on two datasets: Poker Hands and Image Segmentation. For each of the two classification tasks, we tried a plethora of different combinations of value parameters for alpha, momentum, maximum error rate and hidden neurons. Poker Hands dataset gave a minimum mean squared error of 0.00176 for 18 hidden neurons, maximum error 0.01, alpha value 0.1 and momentum value 0.2. For the Image Segmentation dataset, the minimum mean squared error was 0.1122 was obtained for 10 hidden neurons, maximum error of

0.01, alpha value of 0.01 and zero momentum. The full observations are illustrated in the tables.



Training Attempt	Number of Hidden Neuron	Maximum Error	Learning Rate	Momentum	Total Mean Square Root	Number of Iteration	Network Trained
1	10	0.01	0.1	0.1	0.12834	100	yes
2	10	0.01	0.1	0.8	NIL	>1000	no
3	10	0.01	0.2	0.7	0.1439	120	yes
4	10	0.01	0.15	0.01	NIL	>1000	no
5	10	0.01	0.3	0.4	0.13695	5	yes
6	20	0.01	0.6	0.1	0.012393	590	yes
7	20	0.01	0.7	0.2	NIL	>1000	no
8	20	0.01	0.8	0	0.1403	19	yes

Figure 5.1: Various parameters of Multi-Layer Perceptron and root square mean error for the Image Segmentation Dataset.



Training Attempt	Number of Hidden Neuron	Maximum Error	Learning Rate	Momentum	Total Mean Square Root	Number of Iteration	Network Trained
1	5	0.01	0.2	0.7	NIL	>8000	no
2	5	0.01	0.5	0.7	NIL	>5000	no
3	10	0.01	0.2	0.7	NIL	>3300	no
4	10	0.03	0.2	0.7	0.00738	134	no
5	12	0.02	0.2	0.7	0.0055	224	no
6	12	0.01	0.1	0.7	0.00198	185	yes
7	15	0.01	0.2	0.7	0.00225	152	yes
8	15	0.01	0.1	0.8	0.00246	121	no

Figure 5.2: Various parameters of Multi-Layer Perceptron and root square mean error for the Poker-Hand Dataset.

5.0.3 ARTMAP

For ARTMAP, we used the Java port of the original ARTMAP source code provided by Cognitive and Neural Systems Technology Laboratory at Boston University, by the inventors of the algorithm, Carpenter and Grossberg. Our first test run was on the relatively simple IRIS dataset, consisting of 4 input feature vectors and 3 output classes. The algorithm correctly classified 147/150 test cases correctly, thereby, giving an accuracy of 98%. For the Poker Hands database, the algorithm could classify 7,12,345 out of 10,00,000 test cases correctly, giving an accuracy of 72.1%. For the Image Segmentation database, ARTMAP gave an accuracy of 100% , classifying all 210 test cases correctly.

5.0.4 LEARN++

For LEARN++, we used a family of binary CART Decision Tree classifiers. Being a binary classification model, we used the Ionosphere dataset, consisting of 351 instances of 34-dimensional input data. We tested the code for several values of K (number of input segmentations) and number of iterations. It gave a minimum error of 0.0176 and maximum error of 0.3429. The average values are plotted and are illustrated in the graph.

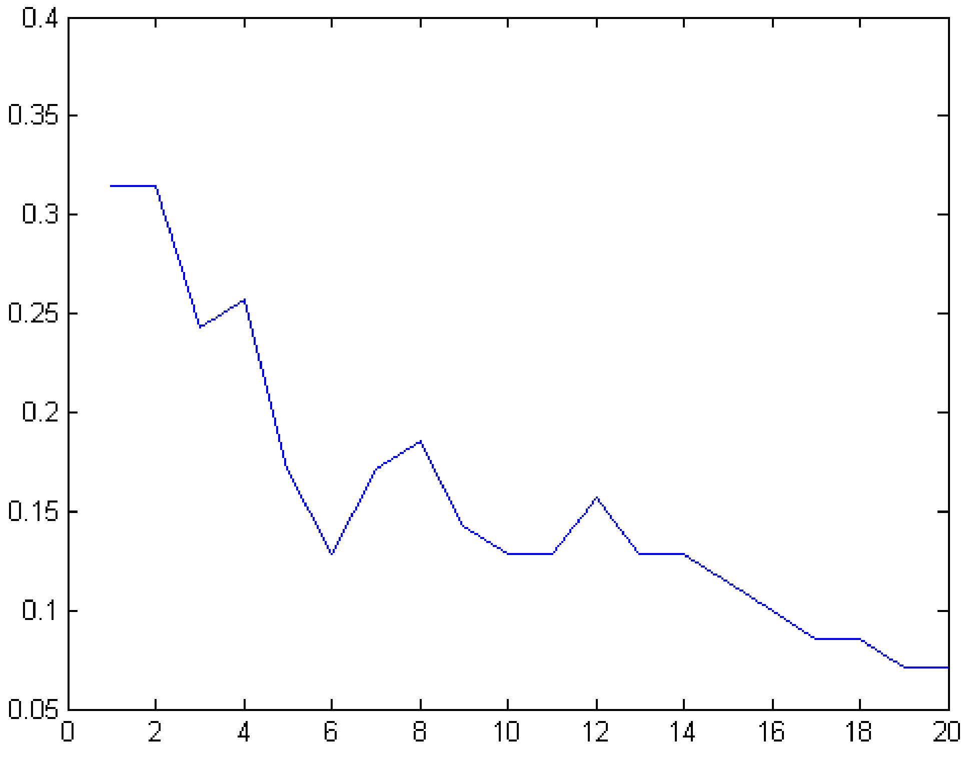
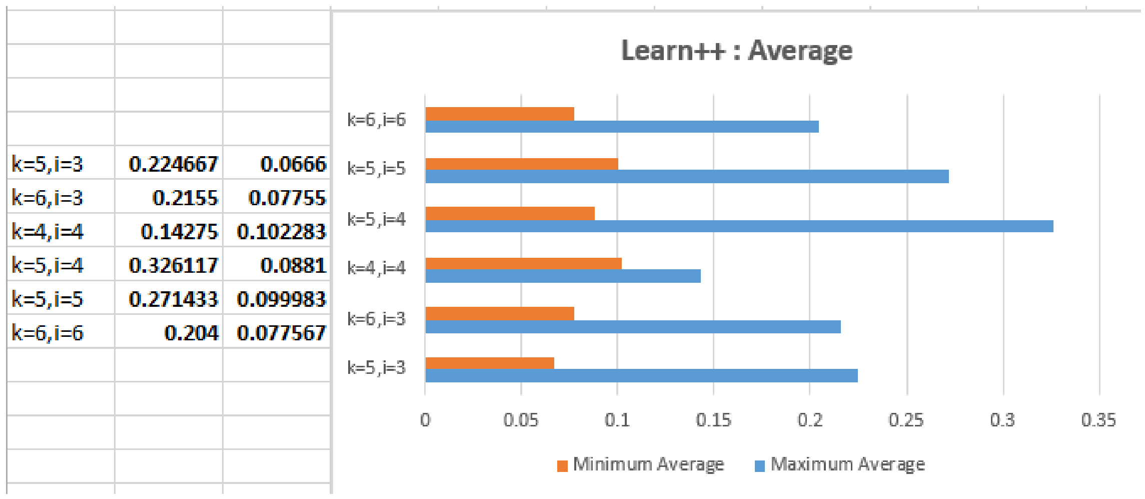


Figure 5.3: Error Vs Iteration Plot for LEARN++ Data Parts : 5 Iterations : 4



k=5,i=3	0.224667	0.0666
k=6,i=3	0.2155	0.07755
k=4,i=4	0.14275	0.102283
k=5,i=4	0.326117	0.0881
k=5,i=5	0.271433	0.099983
k=6,i=6	0.204	0.077567

Figure 5.4: LEARN++: Plot of min and max error for various iterations

Chapter 6

Conclusion

6.0.5 Face-off : Backpropagation vs. ARTMAP

Both Multi-layer perceptrons and ARTMAP are Neural-Network based architectures, which are aimed at Pattern learning, recognition and classification. However, both the architectures are fundamentally different, as ARTMAP allows for Incremental Learning, whereas Backpropagation results in Catastrophic Forgetting. In other words, an MLP can not learn in batches, whereas ARTMAP can. As far as raw performance is concerned, our observations were quite unique and unprecedented. We observed that on small datasets, both the approaches gave near perfect results (98-99%). However, there are a lot of variations when it comes to medium and large sized datasets. ARTMAP didnt perform well for Poker Hands, giving an accuracy of 72.1%, whereas Backpropagation gave a mean squared error of 0.00176 in the best possible case and a maximum error of 0.01, which makes it the clear winner in this case. However, since ARTMAP has only two layers of neurons, it is significantly faster.

6.0.6 Face-off : ARTMAP vs. LEARN++

The inventor of LEARN++ claims that it is better than ARTMAP in several respects. In many applications, researchers have noticed that ARTMAP is very sensitive to selection of the vigilance parameter, to the noise levels in the training data and to the order in which the training data is presented to the algorithm. Furthermore, the algorithm generates a large number of clusters causing overfitting, resulting in poor generalization performance, if the vigilance parameter is not chosen correctly. Therefore, this parameter is typically chosen in an ad hoc manner by trial and error. In terms of performance, both ARTMAP and LEARN++ perform exceedingly well for a dual-class classification, giving 97-99% accuracy for small-to-medium datasets. Changing the k and i values in LEARN++ leads to different observations.

6.0.7 Face-off : Backpropagation vs LEARN++

Backpropagation and LEARN++ have a lot of dissimilarities to note. To start off, Backpropagation is a Neural-network approach, whereas LEARN++ is an ensemble learning approach to pattern classification. Backpropagation is not incremental in nature, whereas LEARN++ is. However, another point worthy of attention is that a BPNN may itself be a part of the ensemble methods that comprise LEARN++. The success of LEARN++ is attributed to the success of its ensemble methods, as well as the AdaBoost algorithm. Back propagation gave an average error of 3.3% on the Ionosphere dataset. In our observation, LEARN++ gave a better result by as it gave an minimum error of 1.7%.

Chapter 7

Discussion and Future Work

ART networks serve both as models of human cognitive information processing and as neural systems for technology transfer. Design principles derived from scientific analyses and design constraints imposed by targeted applications have jointly guided the development of many variants of the basic networks, including fuzzy ARTMAP, ART-EMAP, ARTMAP-IC, and Gaussian ARTMAP. Early ARTMAP systems, including fuzzy ARTMAP, employ winner-take-all (WTA) coding, whereby each input activates a single category node during both training and testing. When a node is first activated during training, it is mapped to its designated output class. Starting with ART-EMAP, subsequent systems have used distributed coding during testing, which typically improves predictive accuracy, while avoiding the computational problems inherent in the use of distributed code representations during training. In order to address these problems, distributed ARTMAP introduced a new network configuration, in addition to new learning laws.

Comparative analysis of the performance of ARTMAP systems on a variety of benchmark problems has led to the identification of a default ARTMAP network, which features simplicity of design and robust performance in many application domains. Default ARTMAP employs winner-take-all coding during training and distributed coding during testing within a distributed ARTMAP network architecture. With winner-take-all coding during testing, default ARTMAP reduces to a version of fuzzy ARTMAP.

ART and ARTMAP employ a preprocessing step called complement coding, which models the nervous systems ubiquitous use of the computational design known as opponent processing. Balancing an entity against its opponent, as in against-antagonist muscle pairs, allows a system to act upon relative quantities, even as absolute magnitudes may vary unpredictably. In ART systems, complement coding is analogous to retinal ON-cells and OFF-cells. When the learning system is presented with a set of input features, complement coding doubles the number of input components, presenting to the network both the original feature vector and its complement.

Complement coding allows an ART system to encode within its critical feature patterns of memory features that are consistently absent on an equal basis with features that are consistently present. Features that are sometimes absent and sometimes present when a given category is learning become regarded as uninformative with respect to that category. Since its introduction, complement coding has been a standard element of ART and ARTMAP networks, where it plays multiple computational roles, including input normalization. However, this device is not particular to ART, and could, in principle, be used to preprocess the inputs to any type of system.

References

[1] Artmap module. `http://http://techlab.bu.edu/classer/artmap_module`.

[2] Gail A Carpenter, Stephen Grossberg, and John H Reynolds. Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural networks*, 4(5):565–588, 1991.

[3] Hussein Syed Mohammed, James Leander, Matthew Marbach, and Robi Polikar. Can adaboost. m1 learn incrementally? a comparison to learn++ under different combination rules. In *Artificial Neural Networks–ICANN 2006*, pages 254–263. Springer, 2006.

[4] Robi Polikar, Lalita Udpa, Satish S Udpa, and Vasant Honavar. Learn++: an incremental learning algorithm for multilayer perceptron networks. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 6, pages 3414–3417. IEEE, 2000.