2015 IEEE 15th International Conference on Data Mining Workshops

HARAM: a Hierarchical ARAM neural network for large-scale text classification

Fernando Benites, Elena Sapozhnikova
Department of Information and Computer Science,
University of Konstanz
{Fernando.Benites,Elena.Sapozhnikova}@uni.kn

***Abstract*—With the rapid development of the web, the need for text classification of large data volumes is permanently growing. Texts represented as bags-of-words possess usually very high dimensionality in the input space and often also in the output space if labeled with many categories. As a result, neural classifiers should be adapted to such large-scale data. We present here a well scalable extension to the fuzzy Adaptive Resonance Associative Map (ARAM) neural network which was specially developed for high-dimensional and large data. This extension aims at increasing the classification speed by adding an extra ART layer for clustering learned prototypes into large clusters. In this case the activation of all prototypes can be replaced by the activation of a small fraction of them, leading to a significant reduction of the classification time. This extension can be especially useful for multi-label classification tasks.**

I. INTRODUCTION

With the rapid development of the web, the amount of electronic documents in digital libraries is dramatically increasing nowadays. Consequently the need for text classification of large data volumes is ever increasing. The high dimensionality of such data, concerning both input and output spaces, is one of the most important issues in modern text classification problems. To cope with this challenge, the scalability of existing classification algorithms should be improved. We present here an extension of fuzzy Adaptive Resonance Associative Map (ARAM) [1] – an Adaptive Resonance Theory (ART)-based neural network. It aims at speeding up the classification process in the presence of very large data.

ART neural networks belong to the class of competitive learning networks with a growing structure [2], creating new prototypes on-the-fly during the learning phase. The most important advantages of Fuzzy ART networks, Fuzzy ARTMAP (FAM) and ARAM, are their fast learning ability inherent in ART [3], simple extraction of the learned fuzzy rules, the ease of parameter setting, debugging and implementation. Their prototypes represent input data by multi-dimensional hyperboxes. The hyperbox size is bounded above by a user-defined threshold parameter ρ called vigilance. The greater the vigilance, the smaller the hyperboxes and more specific the prototypes. There is however no hint how to set this parameter in a certain classification task. Moreover the influence of vigilance on the hyperbox size depends on the dimensionality of a dataset because of the definition of the size as the sum over all dimensions (see Eq. (7)). Thus the higher the dimensionality, the finer should be tuned the vigilance. Consequently, increasing this parameter by only 0.00001 can dramatically increase the number of prototypes in a high-dimensional setting.

It is known that the efficiency of ART networks can decrease due to building too many prototypes, which is a problem of category proliferation [4]. One of its reasons can be overtraining on large and overlapping data [5], [6], in particular due to the match tracking procedure [3]. In this case, the growth of the network can become a problem at the test stage because for each sample to be classified a huge number of prototypes must be activated. For example, in an earlier experiment on single-label text classification [7] ARAM created more than 2700 prototypes after training on only 7770 documents with $\rho = 0.8$.

The problem becomes even more pronounced in multi-label setting because multi-label classifiers ML-FAM and ML-ARAM [8] process each multi-label as a unique class that leads to more invocations of the match tracking procedure. Although both classifiers showed performance superior to the non-neural multi-label classifiers [9] and comparable with other neural multi-label classifiers (see Table IV), we expect deceleration of the classification process for multi-label and large datasets.

To solve the problem of activation of many prototypes, spatial indexing methods such as k-d Tree [10], R-Tree [11] or X-Tree [12] can be applied. The idea of these methods is the hierarchical organization of data into a tree structure using the feature values in order to accelerate access to it. However these methods are either too inaccurate for classification or slower than the activation of all prototypes of ML-ARAM, especially in classification problems with thousands of features like text mining.

For this reason, we propose clustering learned prototypes by means of additional unsupervised Fuzzy ART layers. If the cluster size is chosen sufficiently large, each cluster can then be responsible for multiple prototypes, providing a hierarchical structure with a few clusters. Thus the choice of one cluster will lead to activating only a subset of all prototypes and accelerating the classification process. In this way Hierarchical ARAM (HARAM) can decrease the number of activated prototypes for each test sample in order to shorten activation time as compared with the standard (ML)-ARAM algorithm.

HARAM, conventional ARAM and their multi-label extensions were evaluated on popular text datasets. Obtained experimental results were also compared with the state-of-the-art multi-label neural classifiers from [13]. The rest of the paper is organized as follows: After introducing HARAM in

978-1-4673-8493-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDMW.2015.14

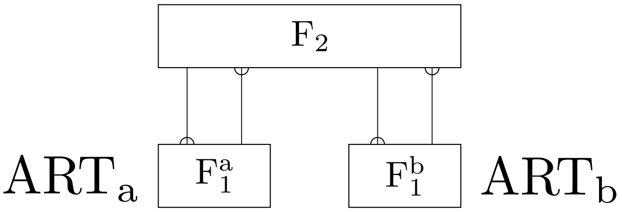

Figure 1. ARAM neural network.

Section II, its multi-label extension ML-HARAM is presented in two versions (Section III). Experimental results on a single-label and two multi-label datasets are discussed in Section IV. Finally, some concluding remarks are given in Section V.

II. HARAM

First of all, the basic steps of the original ARAM algorithm are presented below, for more details see [1]. ARAM performs incremental supervised learning of pattern pairs. It can be visualized as two overlapping Fuzzy ART [2] modules $\mathrm{ART_a}$ and $\mathrm{ART_b}$ consisting of two input fields $\mathrm{F_1^a}$ and $\mathrm{F_1^b}$ connected through weights to a common category field F_2 (Figure 1). The network processes complement-coded [2] inputs $\boldsymbol{A}$ and targets $\boldsymbol{B}$ by adapting existing prototypes or creating new ones, which are stored in the weights of category nodes of the F_2 layer. In classification tasks, the target vectors usually represent class labels, for example, in the binary form. When performing a classification task, ARAM incrementally creates prototypes for input patterns, and associates each prototype with its respective class.

At the beginning of the training process, all weight vectors $\boldsymbol{W}_k$ ($k = 1, \ldots, N$, where N is the number of category nodes, initially $N = 1$) are set to unity and the nodes are said to be uncommitted. After the presentation of $\boldsymbol{A}$, the activation function T_k is calculated and the winner K is chosen by the Winner-Take-All (WTA) rule (2):

$$T_k(\boldsymbol{A}) = \frac{|\boldsymbol{A} \wedge \boldsymbol{W}_k^{\mathrm{a}}|}{\alpha + |\boldsymbol{W}_k^{\mathrm{a}}|} \tag{1}$$

where $\wedge$ denotes the fuzzy AND, element-wise min operator, and $\alpha > 0$ is called the choice parameter. (This form corresponds to setting the contribution parameter $\gamma = 1$ in the original ARAM.)

$$T_K = \max\left\{T_k : \, k = 1, \ldots, N\right\} \tag{2}$$

Then the winner choice should be confirmed by the pair of so-called match criteria which define the minimum required similarity between the winner's prototype and the input at $\mathrm{ART_a}$ as well as between the actual and previously learned target at $\mathrm{ART_b}$

$$\frac{|\boldsymbol{A} \wedge \boldsymbol{W}_K^{\mathrm{a}}|}{|\boldsymbol{A}|} \geq \rho_{\mathrm{a}}, \qquad \frac{|\boldsymbol{B} \wedge \boldsymbol{W}_K^{\mathrm{b}}|}{|\boldsymbol{B}|} \geq \rho_{\mathrm{b}} \tag{3}$$

where ρ_{a} and $\rho_{\mathrm{b}} \in [0, 1]$ are the respective user-defined vigilance parameters for the minimum accepted similarity.

If the $\mathrm{ART_a}$ inequality of (3) is violated, the node K is inhibited and the network enables another node to be selected. This search process continues until the input is either assigned to a committed node or codes the prototype of a new uncommitted node. However if the $\mathrm{ART_b}$ inequality is violated, i.e. the class mismatch occurs, the so-called match tracking process is started by increasing the vigilance until it becomes slightly higher than the the left-hand side of the $\mathrm{ART_a}$ inequality for the time of the current input presentation. This ensures a correct class prediction in the regions of potential class overlap because match tracking corrects an erroneous prediction for a training point by raising the vigilance and building a new smaller hyperbox of the proper class. So, match tracking stimulates category proliferation by creating finer nested hyperboxes of different classes mostly on the boundaries between classes [14]. A faster alternative with a slightly inferior classification performance would be to activate only the prototypes of a proper class in a supervised fashion without match tracking. Though it can lead to misclassifications in a single-label case, for the multi-label classification it is useful because of acquiring the class information from several prototypes [8]. This modification can accelerate the training process significantly due to the activation of only a fraction of the prototypes, i.e. those that possess the same class as the training sample.

When each match criterion is satisfied in the respective module, the learning process follows. During learning the winner K learns to encode the input and target vectors by adjusting its weight vectors $\boldsymbol{W}_K^{\mathrm{a}}$ and $\boldsymbol{W}_K^{\mathrm{b}}$ accordingly to (4).

$$\begin{aligned} \boldsymbol{W}_K^{\mathrm{a(new)}} &= \beta_{\mathrm{a}} \left(\boldsymbol{A} \wedge \boldsymbol{W}_K^{\mathrm{a(old)}}\right) + (1 - \beta_{\mathrm{a}})\, \boldsymbol{W}_K^{\mathrm{a(old)}} \\ \boldsymbol{W}_K^{\mathrm{b(new)}} &= \beta_{\mathrm{b}} \left(\boldsymbol{B} \wedge \boldsymbol{W}_K^{\mathrm{b(old)}}\right) + (1 - \beta_{\mathrm{b}})\, \boldsymbol{W}_K^{\mathrm{b(old)}} \end{aligned} \tag{4}$$

where β_{a} and $\beta_{\mathrm{b}} \in [0, 1]$ are learning rates, set to unity in the fast learning mode. This process can be seen as the growth of the hyperbox just to include a learned data point, if it is not already included. A graphical illustration of the learning process at $\mathrm{ART_a}$ can be found in Figure 2. The weight vector[1] $\boldsymbol{W}$ contains the minimum point $\boldsymbol{r}^{min}$ and the complement of the maximum point $\boldsymbol{r}^{max}$ of the respective hyperbox. The match criterion of $\mathrm{ART_a}$ corresponds to bounding the maximum size of a new hyperbox created after expansion. The hyperbox size is calculated on the basis of the L_1 norm as the sum of the lengths of its sides:

$$\begin{aligned} |R_j| =& |\boldsymbol{r}_j^{max} - \boldsymbol{r}_j^{min}| = \sum_i \left((1 - W_{i+M,j}) - W_{i,j}\right) \\ =& M - |\boldsymbol{W}_j| \end{aligned} \tag{5}$$

where M is the input dimensionality. From the left part of (3) it follows that

$$|\boldsymbol{W}^{(\mathrm{new})}| \geq \rho |\boldsymbol{A}| = \rho M \tag{6}$$

and correspondingly

$$|R_j^{(\mathrm{new})}| = M - |\boldsymbol{W}^{(\mathrm{new})}| \leq (1 - \rho) M. \tag{7}$$

---

[1] We will skip further the symbol "a" for notational simplicity.

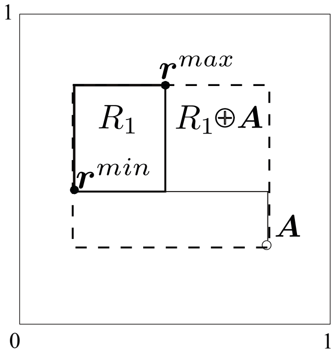

Figure 2. The hyperbox learning process.

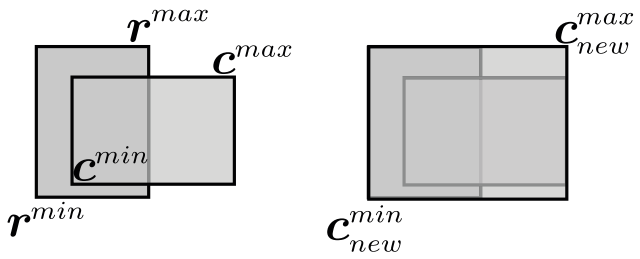

Figure 3. Cluster growth

That is a higher vigilance leads to smaller hyperboxes, but the choice of appropriate values for the vigilance parameter depends on the dataset dimensionality. Even a very small change of the ρ value multiplied by a large number of dimensions like 50000 causes a high variation in the maximum hyperbox size.

HARAM is based on the algorithm described above, but has several important differences aimed at accelerating the classification process. First, only committed nodes take part in the competition in our implementation. In contrast to original ARAM where one uncommitted node is allowed to compete with committed ones, a new node is added only if needed. This helps prevent category proliferation. The second difference is an additional preparation step which takes place after the training process is completed. At this step, higher layers forming the clusters of learned prototypes are organized. They reduce the number of activated F_2 prototypes[2] at the classification step. During clustering of prototypes, their identifiers are stored for a rapid access from the top. In order for this to happen, the learned F_2 prototypes are taken as input for training of an additional unsupervised Fuzzy ART layer. The winner is found similarly to (1) by choosing the cluster with the highest activation:

$$T^{\mathrm{c}}_k(\boldsymbol{W}^{\mathrm{a}}_j) = \frac{|\boldsymbol{W}^{\mathrm{a}}_j \wedge \boldsymbol{W}^{\mathrm{c}}_k|}{\alpha + |\boldsymbol{W}^{\mathrm{c}}_k|} \tag{8}$$

here $\boldsymbol{W}^{\mathrm{c}}_k$ is the weight vector of the cluster and $\boldsymbol{W}^{\mathrm{a}}_j$ the prototype of the network. An important difference is that here not a single complement-coded input point but two hyperbox points ($\boldsymbol{r}^{min}$ and a $\boldsymbol{r}^{max}$) coded in $\boldsymbol{W}^{\mathrm{a}}_j$ are used. So, in contrast to Eq. (1), both points influence the activation value and thus creating compact clusters is rewarded.

The growing process of a cluster C after learning a prototype P is depicted in Figure 3. The corresponding hyperbox with the corner points $(\boldsymbol{r}^{min}, \boldsymbol{r}^{max})$ becomes incorporated in the cluster $C = (\boldsymbol{c}^{min}, \boldsymbol{c}^{max})$ changing it to C_{new}=$(\boldsymbol{c}^{min}_{new}, \boldsymbol{c}^{max}_{new})$. Then if C_{new} is the winner of the higher layer, the prototype P is activated at F_2 along with all other prototypes involved in the creation of C.

The Fuzzy ART clustering layer usually has a vigilance value (which we denote as Clustering Vigilance (CV)) lower than the value chosen for ARAM training. In principle, clustering can be performed in several layers, using prototypes/clusters from a lower layer as the input for the next layer and decreasing the vigilance parameter for each new higher layer. Hierarchical activation starts then by activating the clusters of higher layers and goes down, only activating the prototypes of the lower layer which belong to the winner at the higher layer. So, first the clusters of the highest layer are activated by a test sample. The winning cluster propagates the pointers to the prototypes to be activated at the layer below. After activating only the selected prototypes of this layer, new pointers are restored from the winning cluster and the process continues until the lowest layer F_2 is reached, from which the corresponding classification labels can be taken.

Although this builds a hierarchy of prototypes, the gain of speed decreases significantly with each additional layer and accuracy hardly changes at all. Several experiments not included in this paper due to the space constraints showed that besides the F_2 prototype layer, it was the first cluster layer that influenced the accuracy of the classifier most. For this reason only one cluster layer will be used in the experiments below.

The clustering process creates larger (and most importantly, fewer) clusters and is thereby able to accelerate the access to the prototypes. In order to maximize the gain of speed, one can use a simple rule of thumb as a guiding value for the optimal number of clusters: Assuming the equal distribution of prototypes in clusters, this value can be obtained by minimizing the sum of the number of activated clusters and the number of activated prototypes in the winning cluster which gives the square root of the prototype number.

Although the described process is performed offline, it can be modified in order to retain the valuable online learning property of ART. It is also possible to train the clusters at the same time as the prototypes directly on training samples. However this would cause a much longer training time.

Two potential issues of HARAM, which are related to each other, are the problem of neighborhood and overlapping clusters. In the first case, if a data point is not covered by any prototype, it can still be covered by a cluster. A simple way to visualize this is if we start from a rectangle (Cluster C) and divide it equally into four subrectangles, as can be seen in Figure 4. Two of them, P_i and P_j, on the opposite corners represent prototypes and the other two empty spaces were aggregated by the training process. If a point A lies in these empty spaces farther away, near the border, there might

---

[2] We differentiate here between F_2 prototypes, which are coded in the category nodes and connected to labels, and clusters of higher layers which are not a part of the ARAM network.

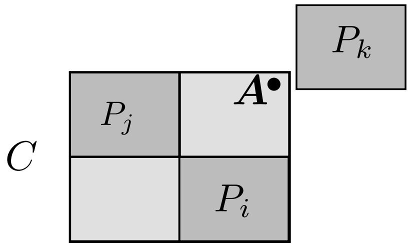

Figure 4. Potential issue of HARAM

be another prototype P_k just outside of C which could be closer to this point than both prototypes in C. So the cluster may create borders between the prototypes which may not be optimal. The other issue, which can also be seen in Figure 5, concerns when a point lies in the region of overlap, where multiple clusters have the maximum activation value at the same time. To overcome both problems we propose to use not the WTA rule of Eq. (2), but three most activated clusters. Thus, in the shown example prototypes of clusters 2 and 4 would be also activated.

Figure 5 shows HARAM handling the well-known problem "the circle in the square" [3]. The ARAM network trained with a vigilance of 0.95 had 125 prototypes. In the upper layer only 5 clusters were created with the CV set to 0.8. In the classification phase with the sample (0.5,0.5), the middle cluster (number 3) won in the higher layer. It contained 30 prototypes that should be activated plus the 42 of its both neighbors. In total, only 77 prototypes from both levels were activated while classifying the sample, taking only about 60% the effort of the standard ARAM network.

This approach differs from the approaches using a hierarchy of ART layers to obtain multi-resolution clustering, e.g. [15], because its main goal is to speed up classification and especially multi-label classification.

III. ML-HARAM

Another important difference between HARAM and ARAM concerns their multi-label versions. ML-ARAM [8] increases its precision by relaxing the WTA rule, allowing multiple prototypes to be involved in the label-ranking calculation. For that it first calculates the difference in activations between the most activated and the least activated prototype. Then this difference multiplied by a user defined ML-threshold is utilized to select a fraction of highly activated prototypes for creating label-rankings. For more details of this process and the subsequent transformation from label-rankings to multi-labels see [8]. In HARAM instead of activating all prototypes, the lowest activation value is estimated in the preparation step: The prototypes are used again for the activation[3], as in the clustering process, but this time only the least activated node is selected for each prototype. The least activated prototype ($P_i^l = \underset{k}{arg\,min} \frac{|\boldsymbol{W}_i^{\mathrm{a}} \wedge \boldsymbol{W}_k^{\mathrm{a}}|}{\alpha + |\boldsymbol{W}_k^{\mathrm{a}}|}$) is chosen and its identifier is saved as an additional attribute of the prototype (P_i) used as input, i.e. we discover the prototype which is the least activated for the given input prototype. The lowest activation value among all neurons can then be estimated from P_i^l, given a certain test sample for which P_i has the highest activation. This strategy, Precalculation of least Activated (PA), is imprecise but very fast. However the error is low comparing with the use of a precise value of the least activated prototype.

IV. EXPERIMENTS

We present here a comparison of (ML)-HARAM with (ML)-ARAM and other classifiers. We implemented the algorithms in Python with the Scipy package. The sparse version, which is well-suited for very large data both concerning its execution time and memory consumption, is implemented in cython. We also compared the "full" and the "sparse" versions of HARAM to demonstrate that the float precision can influence some results, and how they perform in execution time against each other. On the multi-label datasets we compared additionally the standard algorithms with the accelerated algorithms implemented without match tracking (denoted by the suffix $_{wom}$).

For comparison we also used a linear SVM classifier [16] as implemented in http://www.csie.ntu.edu.tw/~cjlin/liblinear. Being fast classifiers, SVMs still have several disadvantages, e.g. black-box nature and only offline learning. Additionally, in multi-label classification tasks they use the Binary Relevance (BR) approach [17] and therefore their parallelization is limited. For the datasets 20 Newsgroups and Reuters we used L2-regularized logistic regression (primal) kernel, but for the EUR-Lex dataset – L2-regularized L2-loss support vector regression (dual), in order to achieve the results comparable with those of [13].

A. Data

First, a single-label text classification was performed on the 20 Newsgroups dataset downloaded from http://qwone.com/~jason/20Newsgroups/ in January 2015. We used the bydate version [18]. The data from 20 topics was already preprocessed and converted in 53975 tf-idf features. We further normalized it column-wise and splitted in a training set containing 11269 samples and a test set with 7505 samples.

We also used the multi-label dataset RCV1-v2 in two versions. The first version is referred here to as Reuters-Small and based on our own preprocessing of the original data provided by http://trec.nist.gov/data/reuters/reuters.html. We used 5000 most frequent features (preprocessed with stop words removal and stemming) in the training set, with tf-idf weighting and normalized them column-wise separately for training and test data. We splitted the original training data into a training and a test set with the ratio of 9:1, e.g. about 21k training samples and 2k test samples.

The second version of the RCV1-v2 dataset is called Reuters-Large dataset later on (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multilabel/ of December 2014) and has 47236 features, 23149 test samples, 781265 training samples and 103 classes. We also swapped the training set and the test set (about 780k for training and 23k for test) as in [13] in order to achieve result comparability. Their results are also similar to those of http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/multilabel/.

---

[3] Here hierarchical activation with clusters and then prototypes can be used to speed up the process.

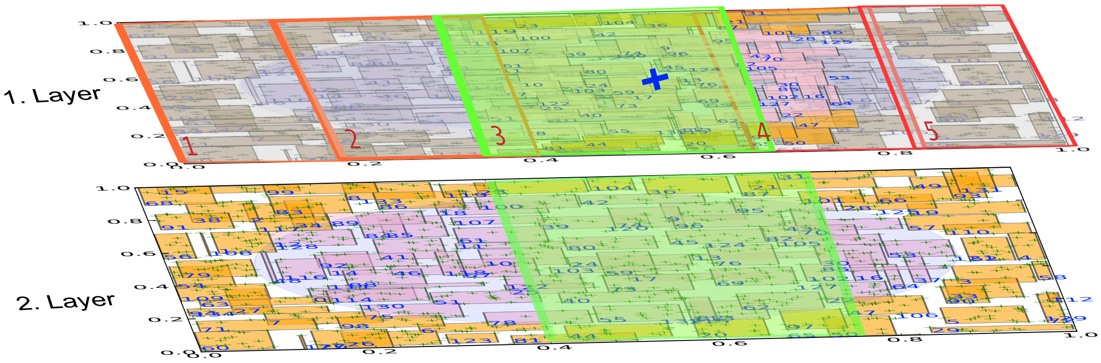

Figure 5. HARAM

We further downloaded the EUR-Lex dataset from http://mulan.sourceforge.net/datasets-mlc.html, more specifically the one with Eurovoc labels for comparison with the results of [13]. It consists of 19348 samples, divided into 17381 training and 1933 test samples with 5000 features and 3993 labels (we used just the first slice of the cross validation). We also performed cosine normalization and removed entries which had no labels, since they created problems while training.

The time was measured on a computer equipped with an Intel(R) Core TM i7-2600 CPU @ 3.40GHz with 32 Gigabytes memory and the bus running at 1.3 GHz (0.8ns). The datasets used here were only a testbed and demonstrated the scalability of the approach. Comparisons and extensive tests in very large datasets would overtop the resources available for this research.

B. Performance measures

The classification performance measure in the single-label case is accuracy which is the ratio of the right classified samples to the total number of classified samples. For multi-label datasets we used the F-1 measure, which is the harmonic mean of recall and precision. It can be calculated in several ways depending on averaging [17]. First, we used instance-based averaging, i.e. we calculated F-1 for every single instance and then took the mean value (denoted as IF-1). As the second measure we utilized the micro-averaged F-1 (denoted as mF-1), i.e. we counted how many true positives (tp), false positives (fp) and false negatives (fn) there were for all labels in the whole test set and then calculated mF-1=$\frac{2*tp}{2*tp+fp+fn}$. Moreover the macro-averaged F-1 MF-1=$\frac{1}{Q}\sum_{i=1\ldots Q}\frac{2*tp_i}{2*tp_i+fn_i+fp_i}$, was used. Here Q is the number of labels and tp_i, fp_i and fn_i are, respectively, the number of true, false positives and false negatives for a label i.

C. Implementation

To cope with large data and to compare with fast implementations, following modification of the standard ARAM was undertaken.

1) Sparse Activation: A general issue with Fuzzy ART networks, is that their complement-coded normalization leads to double-length weight vectors. So, if feature vectors possess high dimensionality, it will cause a large memory consumption. One way to solve this problem is to represent the feature vectors using only the indices and values of the non-zero features. In this case we are interested in increasing the number of features which are equal to zero and therefore replace the complement-coded representation of weight vectors with the representation by the minimum and the maximum points of a hyperbox $\boldsymbol{r}^{min}$ and $\boldsymbol{r}^{max}$. The sparse activation TS is then calculated by taking the minimum between an input vector $\boldsymbol{A}$ and $\boldsymbol{r}^{min}$ plus the ones of the complement and minus the maximum between $\boldsymbol{A}$ and $\boldsymbol{r}^{max}$ as follows:

$$\begin{aligned}TS_k(A) &= \sum_i min(A_i, W_{i,k}) + \sum_i min(1-A_i, W_{i+M,k})\\ &= \sum_i min(A_i, r^{min}_{i,k}) + \sum_i min(1-A_i, 1-r^{max}_{i,k})\\ &= \sum_i min(A_i, r^{min}_{i,k}) + M - \sum_i max(A_i, r^{max}_{i,k})\end{aligned}$$

Further the $\sum_i min(A_i, r^{min}_{i,k})$ can be calculated using only the indices of the non-zero values in A_i and $r^{min}_{i,k}$, since all other features will be zero. $\sum_i max(A_i, r^{max}_{i,k})$ can also be simplified by saving the calculation of $SU = \sum_i r^{max}_{i,k}$, taking into account only the cases when $A_i > r^{max}_{i,k}$, and replacing the value of r^{max}_{ki} by A_i.

$$\sum_i max(A_i, r^{max}_{ki}) = \sum_i r^{max}_{ki} - \sum_{i\in\{j|A_j>r^{max}_{kj}\}} A_i - r^{max}_{ki}$$

This removes all but a fraction of the original calculations, which is left to be done at each test iteration. An important condition for the sparse activation to be faster than the standard one is that $\boldsymbol{r}^{max}_{\boldsymbol{k}}$ has (much) more non-zero features than the input vector $\boldsymbol{A}$.

D. Results

1) 20 Newsgroups: First, we compared the classifiers on the 20 Newsgroups dataset. The algorithms were trained once and run multiple times to see the difference in the test time. As can be seen from Table I, ARAM and HARAM produced similar results in terms of performance measures. The test time of HARAM was significantly shorter than that of the standard algorithm.

In Table I, the voting strategy with five voters[4] increased the performance of all algorithms (only the average time

---

[4] Each voter was trained with a different presentation order of the training set.

Table I. RESULTS OBTAINED ON 20 NEWSGROUPS, WITH THE TEST TIME AND TTPS: TEST TIME PER SAMPLE IN SEC. THE TEST TIME FOR ONE NETWORK WAS AVERAGED OVER 5 TRIALS. THE TEST TIME FOR VOTING IS THE AVERAGED TIME OF A SINGLE VOTER. THE S STANDS FOR SPARSE. CLUSTERING VIGILANCE IS IN BRACKETS.

Classifier	test time (s)	TTpS	Accuracy	Neurons/Clusters
ARAM	4084.22±51.61	0.5442	0.733	1124
ARAM$_S$	633.66 ±0.96	0.0844	0.733	
		HARAM		
[-,0.8]	1589.50±146.27	0.2118	0.725	1124/8
S [-,0.8]	312.84 ±0.79	0.0417	0.725	1124/8
S [-,0.95]	160.87±1.23	0.0214	0.732	1124/26
		5 voters		
ARAM	3530.67±523.93	0.4704	0.740	964±147.5
ARAM$_S$	573.69±74.48	0.0764	0.740	
		HARAM		
[-,0.8]	1168.69±209.63	0.1557	0.734	964±147.5/8±0.4
S [-,0.8]	280.95±35.68	0.0374	0.734	964±147.5/8±0.4
S [-,0.95]	163.76±4.51	0.0218	0.740	964±147.5/ 25±1.0

Table II. RESULTS OBTAINED ON REUTERS-SMALL WITH 23149 SAMPLES DIVIDED AS 9/1, TTPS: TEST TIME PER SAMPLE IN SEC; THE INDEX wom STANDS FOR WITHOUT MATCH TRACKING. VG STANDS FOR VIGILANCE. ML-THRESHOLD WAS 0.02. VIGILANCES ARE IN BRACKETS.

Classifier [vg{,CV}]	IF-1	mF-1	MF-1	TTpS	Neurons/Clusters
		ML-ARAM			
[0.9]	0.757	0.736	0.360	0.237	5590
wom [0.9]	0.791	0.774	0.419	0.048	1553
[0.975]	0.813	0.797	0.497	0.302	7118
wom [0.975]	0.837	0.822	0.553	0.067	2230
		ML-ARAM Sparse			
[0.975]	0.814	0.798	0.495	0.290	7049
wom [0.975]	0.836	0.823	0.552	0.123	2230
		ML-HARAM			
[0.975,0.9]	0.791	0.770	0.482	0.015	7118/133
wom [0.975,0.9]	0.816	0.797	0.521	0.010	2230/139
wom [0.975,0.95]	0.822	0.803	0.542	0.014	2230/287
wom [0.975,0.975]	0.830	0.812	0.541	0.020	2230/500
		ML-HARAM Sparse			
[0.975,0.9]	0.787	0.770	0.473	0.023	7049/132
wom [0.975,0.9]	0.816	0.800	0.524	0.021	2230/139
wom [0.975,0.95]	0.821	0.805	0.541	0.031	2230/287
wom [0.975,0.975]	0.829	0.812	0.539	0.041	2230/500

values for a single voter are depicted, since voting can be parallelized). The difference between the measured time of multiple voters and that of running one network multiple times comes from many factors, e.g. the garbage collection of Python, other concurrent processes, disk usage etc. It can be useful to see how much variation can be expected from this implementation. The differences in the test times of ARAM and HARAM are still so significant that such oscillations do not have a remarkable impact on the comparison.

The sparse implementation was much faster in this experiment, since the number of features in this dataset is high (53975) and thus the sparse activation has a tremendous advantage against the dense algorithms, between 2 and 7 times faster, even for HARAM. For the other datasets (Reuters-Small and EUR-Lex) with far fewer features (5000), this is not true.

An interesting issue which we left for future work, would be to investigate how the presentation order of the learned prototypes influences the clustering process, and therefore the test time and accuracy.

2) Reuters: In Table II we compare ML-ARAM and ML-HARAM as well as their sparse implementations on the Reuters-Small dataset. We varied the CV parameter and used the modification without match tracking (denoted as wom), since this dataset is multi-label. Comparing the results of the standard and modified algorithms, it is obvious that match tracking causes too many neurons to be created without any performance improvement. As it has been discussed earlier, one can see that in the multi-label context it does not achieve the goal of increasing classification performance.

The performance measures micro F-1 and instance-based F-1 attested to similar values of ARAM and HARAM when the CV was high. This parameter has a great impact on the performance measures and TTpS. The simple relation higher the CV, the higher the measures and TTpS, is mostly true. However even with the lower CV of 0.9 and the higher vigilance for the F_2 prototypes (0.975), the performance measures and TTpS of ML-HARAM were better than those of ML-ARAM using a vigilance of 0.9. Furthermore, its test time was about 15 times shorter in terms of TTpS (0.015 against 0.237).

It is important to note that applying the same vigilance value to clustering as used for classification can still improve performance because prototype building is controlled not only by the vigilance parameter but also by their labels. This can be seen from the last rows of the HARAM results. The TTpS in such cases is still much better, about three times better.

ML-HARAM without match tracking, with a vigilance of 0.975 and a CV of 0.9 was, by comparable performance, 30 times faster than the standard ML-ARAM with the same vigilance. Even with higher values of CV it was still up to 15 times faster and achieved better F-1 performance.

The sparse version of ML-ARAM takes about two times longer than the standard ML-ARAM in terms of TTpS on this dataset. It is because the number of features is relatively small (5000) and the selected features were chosen by having a high term frequency, causing the feature vectors to be densely populated. With a higher number of features and sparsely populated vectors, the sparse version would achieve better results, as shown by the 20 Newsgroups dataset.

We also analyzed the difference between the use of a global estimation of the least activated prototype and the precise value of the lowest activation for each test sample. As discussed above this value is needed to calculate how many prototypes should be used in the calculation of rankings and then multi-labels. Activating all prototypes and taking the explicit value of the lowest activation led to a higher mean number of used prototypes (1.58 vs. 1.46) as expected. This did not change the results very much: For a vigilance of 0.975 and a CV of 0.9 the absolute difference in mF-1 was only about 0.001.

Table III summarizes the results for the Reuters-Large dataset. Here only sparse implementations could cope with the amount of data. The data in the full form would occupy 1.6 terabytes with single precision, whereas in the sparse form it requires only about 240 megabytes. Using the sparse representation, there are many ways to calculate the activation of prototypes in ART networks, since the effectivity of the calculation depends on the assumptions made about the sparseness of the data. The method we developed and implemented deals quickly with large data. Further, only the methods without match tracking were fast enough to train the networks in a reasonable amount of time.

On the Reuters-Large dataset ML-ARAM with five voters

Table III. RESULTS OBTAINED ON REUTERS-LARGE, VIGILANCE FOR ML-(H)ARAM WAS 0.995, ML-THRESHOLD=0.00001 (VT = NUMBER OF VOTERS) WITH SPARSE IMPLEMENTATION. A STANDS FOR ML-ARAM$_{wom}$ AND H FOR ML-HARAM$_{wom}$. CLUSTERING VIGILANCE IS IN BRACKETS.

Classifier	IF-1	mF-1	MF-1	test time (s)	Neurons/Clusters
A	0.820	0.800	0.592	48132.03	21863
H [-,0.8]	0.802	0.790	0.536	8528.62	21863/18
H [-,0.95]	0.780	0.762	0.516	3765.89	21863 /318
7 Cluster					
H [-,0.95]	0.802	0.781	0.556	4490.40	21863 /318
5 voters					
A	0.833	0.816	0.609	50782.1±34.2	22884.4±2.4
H [-,0.8]	0.826	0.811	0.579	9557.9±32.8	22884.4/22±0
H [-,0.95]	0.812	0.800	0.567	7742.7±40.2	22884.4/363.2±0.5
SVM	0.859	0.852	0.656	9589.0	

Table IV. RESULTS OBTAINED ON REUTERS-LARGE AND TAKEN FROM [13].

Classifier	mF-1	MF-1
Reuters-Large		
NNA	0.8385	0.6457
NNAD	0.8397	0.6404
BP-MLLTA	0.7154	0.4855
BP-MLLTAD	0.6874	0.4483
BP-MLLRA	0.7889	0.5823
BP-MLLRAD	0.7809	0.5694
BRB	0.8533	0.6842
BRR	0.8476	0.6923

has a slightly lower performance in terms of all F-measures than SVM. The performance of ML-HARAM is even lower but its test time is much shorter: ML-HARAM is about ten times faster. Additionally, we can state that the classification performance of both ML-ARAM and ML-HARAM is consistent with the recent results of other neural networks obtained on this dataset [13] and presented in Table IV. The neural networks used there were variations of multi-label Backpropagation (denoted in the table as BP-MLL) as well as single-layer models with the elements of deep learning (NNA and NNAD). We could also reproduce to a large extent the binary-relevance SVM results shown there (BRB and BRR), the minor variations are due to the parameter optimization on a validation set which we did not perform.

One interesting result here is that with a higher CV the network was faster but did not achieve better prediction. With a CV equal to 0.8, 18 clusters divide about 22000 prototypes into very large groups. Although the activation of few clusters is fast, the activation of the prototype layer is then slow because of roughly thousand prototypes in a cluster[5]. On the other hand, with a CV of 0.95 there are 318 clusters with roughly 69 prototypes in each cluster. This makes the activation of the prototype layer faster and explains the time difference of the test phases. The issue with the decrease in classification performance is more complicated but there is a strong indication that the clusters did not divide the prototypes well and are so small that accessing the right cluster might be influenced by noise. Using more clusters to gather the prototypes to be activated can increase the probability of accessing the proper prototypes. This can be seen from Table III: Using 7 instead of 3 most activated clusters with CV=0.95 increases the classification performance to the level of CV=0.8 but the test time remains much shorter (about half).

Table V. RESULTS OBTAINED ON EUR-LEX: ML-(H)ARAM, ML-THRESHOLD 0.0001, VG=PROTOTYPE VIGILANCE. THRESHOLD FOR SVM WAS 0.5. VIGILANCES ARE IN BRACKETS.

Classifier	IF-1	mF-1	MF-1	TTpS	Neurons/Clusters
ML-ARAM					
[0.975]	0.337	0.327	0.116	0.944	16572
wom [0.975]	0.355	0.355	0.118	0.704	15001
wom [0.99]	0.359	0.359	0.118	0.722	15002
wom [0.999]	0.426	0.414	0.148	0.802	16688
ML-ARAM 5 voters					
wom [0.999]	0.427	0.413	0.148	0.85±0.28	16688±0.71
ML-HARAM$_{wom}$ vg=0.975					
[-,0.975]	0.385	0.381	0.130	0.029	15001/522
ML-HARAM$_{wom}$ vg=0.999					
[-,0.975]	0.431	0.436	0.156	0.035	16688/587
[-,0.99]	0.450	0.454	0.163	0.075	16688/1843
ML-HARAM$_{wom}$ 5 voters					
[-,0.99],	0.489	0.455	0.172	0.072±0.05	16688±0.7/1843.0±1.4
SVM	0.511	0.548	0.17	1.109	

3) EUR-Lex: Table V summarizes the results of ML-ARAM and ML-HARAM on the EUR-Lex-Eurovoc. We also compared different implementations of ML-(H)ARAM. The sparse implementation did not improve the time values here since the number of features was limited to 5000. Without match tracking the number of created neurons in ML-ARAM decreased from 16572 to 15001 by the same vigilance and improved performance. This points to overfitting when using match tracking. SVM was much better in terms of most performance measures but slower and had lower MF-1 than ML-HARAM on this dataset. Although it is generally very fast due to the large number of labels and BR approach it had to evaluate 3993 models at each sample.

ML-HARAM outperformed ML-ARAM, by the same vigilance in terms of all performance measures. Its test time was also much shorter. With a higher vigilance and with a high CV, ML-HARAM was even more superior to ML-ARAM. ML-ARAM without match tracking but with a vigilance of 0.999 had slightly more prototypes than with match tracking and the vigilance of 0.975. However the performance measures in the former case were much better that demonstrates again the problems of match tracking.

In this dataset the number of unique multi-labels was high and therefore a problem encountered with ML-ARAM was that using multiple voters with different presentation orders could not help improve the performance, since the same prototypes were created. This implies also that ML-HARAM had to create a high number of prototypes.

Nevertheless voting greatly improved the results of ML-HARAM. This might be because the input order of the clustering process has considerable impact on accuracy. Clustering seems to have a significant effect, as can be seen from a much higher variance compared with the variance of the number of prototypes. We leave further investigations on the sensitivity of clustering to the presentation order for future work. ML-HARAM with voting had such an improvement in terms of the performance measures that its results had a 70% increase in term of IF-1 and achieved the best MF-1 value.

V. CONCLUSION

We introduced HARAM, an extension to the algorithm ARAM, in order to speed up classification time on high-dimensional and very large datasets. The key idea is to

---

[5] It is an estimated average value

accelerate access to the learned prototypes by clustering. An additional ART layer clusters prototypes during a preparation step after training in order to activate only a small fraction of all prototypes. It significantly decreases the classification time for a single test sample.

The time of the preparation step is negligible with respect to the whole classification time for two reasons: It runs only once and most of the time is needed to calculate the least activated prototypes in the multi-label case. However this step can be also performed on the fly.

In comparison with ARAM, HARAM was accelerated in most setups by a factor of 3 to 10. The HARAM results in terms of performance measures were slightly lower for most experiments. An exception was the EUR-Lex-Eurovoc dataset where the ML-HARAM results were better than those of ML-ARAM. However taking into account the obtained speed-up, it is reasonable to increase performance by using multiple voters. This would be still faster than applying the standard ML-ARAM algorithm.

There are several interesting questions left for our future work. How does presentation order influence the clusters? How can we effectively combine online learning with the clustering step of HARAM, i.e. avoiding the whole calculation of the clusters every time a prototype is changed? How to avoid the loss of classification performance?

VI. ACKNOWLEDGMENTS

This study is part of the DAMIART project supported by the German Research Foundation (Deutsche Forschungsgemeinschaft (DFG)).

REFERENCES

[1] A. Tan, "Adaptive resonance associatiove map," *Neural Networks*, vol. 8, pp. 437–446, 1995.

[2] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system." *Neural Networks*, vol. 4, pp. 759–771, 1991.

[3] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps." *IEEE Trans. on Neural Networks*, vol. 3, no. 5, pp. 698–713, 1992.

[4] E. G. Sanchez, Y. A. Dimitriadis, J. M. Cano, and J. L. Coronado, "MicroARTMAP: use of mutual information for category reduction in fuzzy ARTMAP." in *Proc. of the IJCNN-2000*, 2000, pp. 647–652.

[5] M. Georgiopoulos, A. Koufakou, G. Anagnostopoulos, and T. Kasparis, "Overtraining in fuzzy ARTMAP: Myth or reality?" in *Proc. of IJCNN-2001.*, vol. 2, 2001, pp. 1186–1190.

[6] R. S. P Henniges, E Granger, "Factors of overtraining with fuzzy ARTMAP neural networks." in *Proc. of IJCNN-2005*, 2005, pp. 1075–1080.

[7] A.-H. Tan, "Predictive self-organizing networks for text categorization," in *Advances in Knowledge Discovery and Data Mining*, ser. LNCS, 2001, vol. 2035, pp. 66–77.

[8] E. P. Sapozhnikova, "ART-based neural networks for multi-label classification," in *8th Intl. Symp. on Intelligent Data Analysis*, Lyon, France, 2009, pp. 167–177.

[9] F. Brucker, F. Benites, and E. Sapozhnikova, "Multi-label classification and extracting predicted class hierarchies," *Pattern Recognition*, vol. 44, no. 3, pp. 724 – 738, 2011.

[10] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[11] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, Jun. 1984.

[12] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree: An index structure for high-dimensional data," in *Proc. of the 22th Intl. Conf. on Very Large Data Bases*, ser. VLDB '96, 1996, pp. 28–39.

[13] J. Nam, J. Kim, E. Loza Mencía, I. Gurevych, and J. Fürnkranz, "Large-scale multi-label text classification — revisiting neural networks," in *Machine Learning and Knowledge Discovery in Databases*, ser. LNCS, 2014, vol. 8725, pp. 437–452.

[14] E. Sapozhnikova and W. Rosenstiel, "Afc: Art-based fuzzy classifier," in *Knowledge-Based Intelligent Information and Engineering Systems*, ser. LNCS, 2003, vol. 2774, pp. 30–36.

[15] H.-L. Hung, H.-Y. M. Liao, S.-J. Lin, W.-C. Lin, and K.-C. Fan, "Cascade fuzzy art: a new extensible database for model-based object recognition," in *Visual Communications and Image Processing'96*. Intl. Society for Optics and Photonics, 1996, pp. 187–198.

[16] R. en Fan, K. wei Chang, C. jui Hsieh, X. rui Wang, and C. jen Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[17] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Mining multi-label data," in *In Data Mining and Knowledge Discovery Handbook*, 2010, pp. 667–685.

[18] K. Lang, "Newsweeder: Learning to filter netnews," in *Proc. of the 12th Intl. Conf. on Machine Learning*, 1995, pp. 331–339.