Almost Random Projection Machine with Margin Maximization and Kernel Features

Tomasz Maszczyk and Włodzisław Duch

Department of Informatics, Nicolaus Copernicus University
Grudziądzka 5, 87-100 Toruń, Poland
{tmaszczyk,wduch}@is.umk.pl
http://www.is.umk.pl

Abstract. Almost Random Projection Machine (aRPM) is based on generation and filtering of useful features by linear projections in the original feature space and in various kernel spaces. Projections may be either random or guided by some heuristics, in both cases followed by estimation of relevance of each generated feature. Final results are in the simplest case obtained using simple voting, but linear discrimination or any other machine approach may be used in the extended space of new features. New feature is added as a hidden node in a constructive network only if it increases the margin of classification, measured by the increase of the aggregated activity of nodes that agree with the final decision. Calculating margin more weight is put on vectors that are close to the decision threshold than on those classified with high confidence. Training is replaced by network construction, kernels that provide different resolution may be used at the same time, and difficult problems that require highly complex decision borders may be solved in a simple way. Relation of this approach to Support Vector Machines and Liquid State Machines is discussed.

Key words: Neural networks, machine learning, random projections, liquid state machines, boosting

1 Introduction

Although backpropagation of errors (BP) algorithm [1] has been very useful and is still widely used it has several well-known drawbacks and is not a good candidate for a model of neurobiological learning. Alternative models, such as Leabra, use a combination of Hebbian learning with error-correction, creating sparse, simple representation in their hidden layers [2], but so far they have not been too useful in practical applications to approximation or classification problems. The simplest model that has good biological foundation treats microcircuits in neural minicolumns as a kind of neural liquid that resonates in a complex way when a signal comes [3]. Liquid State Machines [4], and echo state networks [5] are now investigated in the "reservoir computing" field, assuming that a large number of randomly connected neurons form a reservoir providing memory for different aspects of signals [6]. Readout neurons extract from this reservoir stable information in real-time reacting to transient internal states formed by microcircuits in this high dimensional system. Projections into high dimensional space increase the probability of data separation, as proved by Cover [7]. This is the main reason for success of the kernel methods in machine learning [8]. It also agrees with the boosting principle [9], treating each projection as a week classifier.

Following these inspirations we have introduced the almost Random Projection Machine (aRPM) algorithm [10], a single hidden layer constructive network based on random projections that are added only if the new node contains useful information. This was determined by checking if the projection contains an interval of relatively pure (single-class) vectors that could be potentially useful. Such localized linear projections are able to solve highly-non-separable problems [11]. They may be further optimized using projection pursuit algorithms based on Quality of Projected Clusters (QPC) indices [12], but this would increase computational costs of the method. So far transformations used to create hidden nodes in aRPM were based on linear projections followed by non-linear separation of intervals, and filtering based on simple correlation coefficient.

Hidden neurons should maximize information transmission, a principle used in deriving learning rules for spiking networks [13]. One can assume that readout neurons with rich connections to different brain areas should be able to discover useful features correlated with a given tasks if they are already present in the neural reservoir, and Hebbian correlation-based learning should be sufficient to form strong weights. In particular, similarity to already learned objects, captured by various kernels, may be quite useful. Therefore instead of using projections based on input features, kernel-based features should also be added to the hidden nodes. Adding new types of features extends the hypothesis space that simple voting or linear discrimination methods are able to explore. Linear kernels define new features $t(\boldsymbol{x};\boldsymbol{w}) = K_L(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{x}\cdot\boldsymbol{w}$ based on a projection on the $\boldsymbol{w}$ direction, or distance along $t(\boldsymbol{x},\boldsymbol{w})$ line. Gaussian kernels $g(\boldsymbol{x},\boldsymbol{w}) = \exp(-||\boldsymbol{x}-\boldsymbol{w}||/2\sigma^2)$ evaluate similarity between two vectors using weighted radial distance function. Support Vector Machines (SVM) find large margin linear discriminators in these spaces without explicitely constructing the feature space. The final discriminant function is constructed as a linear combination of such kernels, creating in fact a weighted nearest neighbor solution. Each support vector used in a kernel may provide a useful feature, but this type of solution is optimal only for data with particular distributions, and will not work well for example on parity data [14] or other problems with complex logical structure [15].

Creating support features directly instead of using kernels based on support vectors $\boldsymbol{w}$ has some advantages, allowing for judicious design of the feature space, optimization of parameters of individual features, selection of features, increased comprehensibility of solutions. Transfer of knowledge between learning of different tasks can be easily implemented by borrowing good non-linear features from other algorithms (learning from others), for example using features corresponding to fragments of path derived from decision trees [16].

In this paper we focus on margin maximization in the aRPM algorithm. New projections should be added only if they increase correct classification probability of those examples that are either on the wrong side, or are close to the decision border. In the next section aRPM algorithm with margin maximization is formally introduced. Section three presents empirical tests and comparisons with standard machine learning methods, and the last section contains discussion and conclusions.

2 Almost random projections with margin maximization

In essence the aRPM algorithm [10] transforms the input features space $\mathcal{X}$ into the support feature space $\mathcal{H}$ discovering various kinds of useful features. The final analysis in the $\mathcal{H}$ space may be done by any machine learning method. In this paper we shall use only the majority voting as it is the simplest model, and linear discrimination, but once a proper information is extracted other classification methods may benefit from it. The emphasis is thus on generation of new features. If linear discrimination with wide margin is used (linear SVM is a very good choice here), and support features are generated using specific kernel $z(\boldsymbol{x};\boldsymbol{w}) = K(\boldsymbol{x},\boldsymbol{w})$, results will be equivalent to the kernel-based SVM. Creating support feature space in an explicit way does not require $O(n^2)$ operations to calculate the full kernel, which for a large number of vectors n may be quite costly. Mixing features from different kernels, using *a priori* knowledge, or adding interesting features extracted from other models, is quite easy. In this paper we shall use only features generated by random projection and Gaussian kernels, filtered by an index that estimates classification margin.

It is convenient to identify each new candidate feature with a class-labeled hidden node $h(\boldsymbol{x};\boldsymbol{w})$ in a constructive network. Projections on random directions $t(\boldsymbol{x};\boldsymbol{w}_i) = \boldsymbol{x}\cdot\boldsymbol{w}_i$ are especially useful if the direction $\boldsymbol{w} = \boldsymbol{x}/||vx||$ is taken as a point $\boldsymbol{x}$ close to the decision border. Selecting such points may be done in a rough way by starting from projections on a line connecting centers of the classes, and finding the overlapping regions.

Although some projections may not be very useful as a whole, but the distribution of the training data along $t(\boldsymbol{x};\boldsymbol{w}_i)$ direction may have a range of values that includes a pure cluster of projected patterns sufficient large to be useful. For example, in case of parity problems [11, 15] linear projections never separate all vectors labeled as even and odd, but projections on diagonal $[1, 1..1]$ direction show alternating large even or odd pure clusters. One can model posterior probability $\mathcal{P}(C|t_i)$ along the $t_i = t(\boldsymbol{x};\boldsymbol{w}_i)$ direction, separating such clusters using $[t_a, t_b]$ intervals. This creates binary features $b_i(\boldsymbol{x}) = t(\boldsymbol{x};\boldsymbol{w}_i,[t_a,t_b]) \in \{0,1\}$, based on linear projection in the direction $\boldsymbol{w}_i$, restricted to a slice of the input space perpendicular to the w_i direction.

Good candidate feature should cover some minimal number η of training vectors. The optimal number may depend on the data, and the domain expert may consider even a single vector to be a significant exception. Here η has been optimized in CV in the 4-15 range, but simply fixing it to $\eta = \max[3, 0.1NC_{\min}]$ value, where $NC_{\min}$ is the number of vectors in the smallest class, is usually sufficient. This condition avoids creation of overspecific features and is applied to all features. After a new feature has been accepted search for new projection is continued few times (here N_{rep} is simply set to 20), and if no new features is accepted the procedure is stopped and other types of features are generated.

To determine interval $[t_a, t_b]$ that extracts a cluster from projection $t(\boldsymbol{x};\boldsymbol{w})$ sort all projected values and call them $t_1 \leq t_2 \leq ...t_n$. Now count how many vectors, starting from $t_a = t_1$, are from the same class C as vector $\boldsymbol{x}$ projected at t_a; find the first vector t_k on the ordered list that is from a different class $C' \neq C$. If $k-1 \geq \eta$ the interval $[t_a, t_b = (t_k - t_{k-1})/2]$ contains sufficient number of vectors to be included and new candidate feature is created. Start again from $t_a = t_b$ and repeat the procedure to check if more intervals may be created using the same projection.

Even the simplest version of aRPM is thus non-linear, and is capable of solving highly-nonseparable problems [10], posing a real challenge to traditional neural networks and SVMs, for example the parity problem for a large number of bits [14].

The second type of features considered here are based on kernels. While many kernels may be mixed together, here only Gaussian kernels with several values of dispersion σ are used for each potential support vector $g(\boldsymbol{x}; \boldsymbol{x}_i, \sigma) = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}||^2/2\sigma^2)$. Local kernel features have values close to zero except around their support vectors $\boldsymbol{x}_i$. Therefore their usefulness is limited to the neighborhood $O(\boldsymbol{x}_i)$ in which $g_i(\boldsymbol{x}) > \epsilon$. In this neighborhood we should have at least η vectors, otherwise candidate feature will be rejected. To create a multi-resolution kernel features based on a few support vectors, candidate features with large σ are first created, providing smooth decision borders. Then significantly smaller σ values are used to create features more strongly localized. In our case 5 values $\sigma =\in \{2^5; 2^2; 2^{-1}; 2^{-4}; 2^{-7}\}$ have been used.

The candidate feature is converted into a permanent node only if it increases classification margin. In SVM classification margin is simply related to the norm of the weight vector $\boldsymbol{w}$ in the kernel space. It is optimized in a fixed feature space using quadratic programming, reducing the number of support vectors and therefore removing some dimensions from the feature space. In our case we have incremental algorithm, expanding feature space until no improvements are made. Expanding the space should move vectors away from decision border. However, only those vectors that are on the wrong side or rather close to the decision border should be moved in the direction of correctly classified vectors, while the remaining vectors may even move slightly towards the decision border. This idea is similar to boosting [9], with each feature treated as a weak classifier, except that these classifiers have high specificity with low recall. We shall use here a simple confidence measure based on the winner-takes-all principle (WTA), summing the activation of hidden nodes. Projections with added intervals give binary activations $b_i(\boldsymbol{x})$, but the values of kernel features $g(\boldsymbol{x}; \boldsymbol{x}_i, \sigma)$ have to be summed, giving a total activation $A(C|\boldsymbol{x})$ for each class. Although probability of classification $p(C|\boldsymbol{x})$ may be estimated by dividing this value through total activation summed over all classes information about confidence will be lost. Plotting $A(C|\boldsymbol{x})$ versus $A(\neg C|\boldsymbol{x})$ for each vector leads to scatterograms shown in Fig. 1, giving an idea how far is a given vector from the decision border.

In the WTA procedure the difference $|A(C|\boldsymbol{x}) - A(\neg C|\boldsymbol{x})|$ estimates distance from the decision border. Specifying confidence of the model for vector $\boldsymbol{x} \in C$ using logistic function: $F(\boldsymbol{x}) = 1/(1 + \exp(-(A(C|\boldsymbol{x}) - A(\neg C|\boldsymbol{x}))))$ gives values around 1 if $\boldsymbol{x}$ is on the correct side and far from the border, and goes to zero if it is on the wrong side. Total confidence in the model may then be estimated by summing over all vectors, and it should reach n for perfect separation. The final effect of adding new feature $h(\boldsymbol{x})$ to the total confidence measure is therefore:

$$U(\mathcal{H},h) = \sum_{\boldsymbol{x}} (F(\boldsymbol{x};\mathcal{H}+h) - F(\boldsymbol{x};\mathcal{H}))$$

If $U(\mathcal{H},h) > \alpha$ than the new feature is accepted, contributing to a larger margin. Parameter α has been fixed at 0.01, lower values will lead to faster expansion of the feature space. Note that the mislabeled cases that fall far from the decision borders will have large number of votes for the wrong class and thus will fall into the saturation region of the confidence functions $F(\boldsymbol{x};\mathcal{H})$, so they will have little influence on the $U(\mathcal{H},h)$ change. This is in contrast to the mean square error or other such measures that would encourage features pushing mislabeled cases strongly towards the decision border.

To make final decision aRPM with margin maximization uses winner-takes-most mechanism or linear discrimination. We have tested also several other models (Naive Bayes, decision trees, neural networks and nearest neighbor methods) in the final $\mathcal{H}$ space, and results in all cases are significantly improved comparing them to the original feature space. The aRPM algorithm is summarized in Algorithm 1. The initial space $\mathcal{H}$ is created treating the single features x_i in the same way as projections $t(vx)$, without checking confidence measures. In this way simple original features that may be very useful, are preserved (note that they are never used in SVMs).

Algorithm 1 aRPM with margin

```
Require: Fix the values of internal parameters $\eta$, $\alpha$ parameters and the set of $\boldsymbol{\sigma}$ dispersions
    (sorted in descending order).
 1: Standardize the dataset, $n$ vectors, $d$ features.
 2: Set the initial space $\mathcal{H}$ using input features $x_i$, with $i = 1..d$ features and $n$ vectors.
 3: for $k = 0$ to $N_{rep}$ do
 4:    Randomly generate new direction $\boldsymbol{w} \in [0,1]^d$
 5:    Project all vectors on direction $t(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$
 6:    Sort $t(\boldsymbol{x})$ values in ascending order, with associated class labels.
 7:    Analyze $p(t|C)$ distribution to find all intervals with pure clusters defining binary features
       $b_i(\boldsymbol{x};C)$.
 8:    if the number of vectors covered by the feature $b_i(\boldsymbol{x};C) > \eta$ and $U(\mathcal{H},b_i) > \alpha$ then
 9:       accept new binary feature $b_i(\boldsymbol{x})$ creating class-labeled hidden network node.
10:       goto 3
11:    end if
12: end for
13: for $j = 1$ to $n_\sigma$ do
14:    Set the Gaussian dispersion to $\sigma = \boldsymbol{\sigma}_j$.
15:    Create kernel features $g_i(\boldsymbol{x}) = \exp(-||\boldsymbol{x}_i - \boldsymbol{x}||^2/2\sigma^2)$.
16:    if $U(\mathcal{H},g_i) > \alpha$ then
17:       accept new kernel feature $g_i(\boldsymbol{x})$ creating class-labeled hidden network nodes.
18:    end if
19: end for
20: Sum the activity of hidden node subsets for each class to calculate network outputs.
21: Classify test data mapped into enhanced space.
```

aRPM with margin maximization may be presented as a constructive network, with new nodes representing transformations and procedures to extract useful features, and additional layers analyzing the image of data in the feature space created in this way. The algorithm has only a few parameters that in the experiments reported below have been fixed at values given above in this section. Although in tests all vectors have been used as potential Gaussian kernel support features, leading to $O(n^2)$ complexity, in practice only a few such features are accepted.

3 Illustrative examples

The usefulness of algorithm described in this paper has been evaluated on six datasets downloaded from the UCI Machine Learning Repository. A summary of these datasets is presented in Tab. 1. These datasets are standard examples of benchmark type and are used here to enable typical comparison of different learning methods. To compare aRPM with and without margin maximization with other popular classification methods 10-fold crossvalidation tests have been repeated 10 times and average results collected in Table 2, with accuracies and standard deviations for each dataset. Results of Naive Bayes, kNN (with optimized k and the Euclidean distance), SSV decision tree [17], SVM with optimized linear and Gaussian kernels are given for comparison.

Title	#Features	#Samples	#Samples per class	Source
Appendicitis	7	106	85 / 21	[18]
Diabetes	8	768	500 / 268	[18]
Glass	9	214	70 / 76 / 17 / 13 / 9 / 29	[18]
Heart	13	297	160 absence / 137 presence	[18]
Liver	6	345	145 / 200	[18]
Wine	13	178	59 / 71 / 48	[18]
Parity8	8	256	128 even, 128 odd	artificial
Parity10	10	1024	512 even, 512 odd	artificial

Table 1. Summary of datasets

The aRPM-no column gives results of our algorithm without margin optimization [10], and the next two columns with margin, using WTA output and linear discrimination output (calculated here with linear SVM). In all cases adding margin optimization has improved results and linear discrimination in the enhanced space achieves the best results, or at least statistically equivalent. In most cases results are better than linear and Gaussian kernel SVM. The only highly non-separable problem here is the 8 and 10 bit parity that is perfectly handled thanks to the projected binary features even with the WTA output.

Only a few kernel features have been selected in most cases, showing that localized projections are able to provide quite good model of the data. This does not mean that a simpler model would not be created if kernel features will be analyzed before projections. An interesting solution that has not been checked yet is to create first low resolution features, with projections and intervals that cover large number of vectors (large η), followed by large kernels, and than remove vectors that are correctly handled and progressively reduce η to account for the remaining errors. The final model may be then analyzed in terms of general rules and exceptions.

Dataset	Method							
	NB	kNN	SSV	SVM(L)	SVM(G)	aRPM-no	aRPM (WTA)	aRPM(LDA)
Append.	83.1 ± 10.2	87.0 ± 10.6	87.9 ± 7.4	85.1 ± 6.0	85.9 ± 6.4	82.6 ± 9.3	87.7 ± 8.1	88.0 ± 6.7
Diabetes	68.1 ± 2.3	75.2 ± 4.1	73.7 ± 3.8	76.4 ±4.7	75.7 ± 5.9	67.7 ± 4.2	61.2 ± 5.7	76.7 ± 4.4
Glass	68.6 ± 9.0	69.7 ± 7.4	69.7 ± 9.4	40.2 ±9.6	63.2 ± 7.7	65.0 ± 9.9	60.3 ± 8.5	68.9 ± 8.3
Heart	76.5 ± 8.6	82.8 ± 6.7	74.7 ± 8.7	83.2 ±6.2	83.5 ± 5.3	78.3 ± 4.2	80.1 ± 7.5	83.1 ± 4.7
Liver	58.6 ± 3.8	62.6 ± 8.5	68.9 ± 9.7	68.4 ±5.9	69.0 ± 8.4	61.1 ± 5.1	67.5 ± 5.5	72.7 ± 7.9
Wine	98.3 ± 2.6	94.9 ± 4.1	89.4 ± 8.8	96.0 ± 5.9	97.8 ± 3.9	68.6 ± 7.8	94.3 ± 5.8	97.7 ± 4.0
Parity8	28.9 ± 4.6	100 ± 0	49.2 ± 1.0	34.1 ±11.7	15.6 ± 22.7	99.2 ± 1.6	100 ± 0	34.7 ± 3.8
Parity10	38.1 ± 3.3	100 ± 0	49.8 ± 0.3	44.1 ±5.0	45.6 ± 4.3	99.5 ± 0.9	100 ± 0	40.3 ± 2.7

Table 2. 10 x 10 crossvalidation accuracy and variance.

The effect of margin optimization is clearly seen in scatterograms, Fig. 1 and 2. Most vectors that were not classified with high confidence are removed now away from the decision border (which is at the diagonal), activating many units, except for those that cannot be classified correctly. Some vectors that were far from decision border have moved a bit closer to it.

4 Conclusions

Almost Random Projection Machine algorithm [10] has been improved in two ways, by adding selection of network nodes to ensure wide margins, and by adding kernel features. Relations to kernel SVM, Liquid State Machines, reservoir computing, and boosting on the machine learning side, and biological plausibility on the other side, make aRPM a very interesting subject of study. In contrast to typical neural networks that learn by parameter adaptation there is no learning involved, just generation and selection of features. Shifting the focus to generation of new features followed by the winner-takes-all algorithm (or linear perceptron), makes it a candidate for a fast-learning neural algorithm that is simpler and learns faster than MLP or RBF networks. Features discovered solving other tasks may be transfered to new tasks facilitating faster learning. Kernel features may be related to neural filters that evaluate similarity to previous complex stimuli. Although kernels have been recommended in cognitive science for analysis of categorization experiments and behavioral data [19] the view expressed in this paper offers a simpler and more comprehensible explanation. Feature selection and construction, finding interesting views on the data is the basis of natural categorization and learning processes.

Results on benchmark problems show improvements of aRPM(LDA) over all other classifiers over the most cases, and results of aRPM(WTA) on the parity problem (and

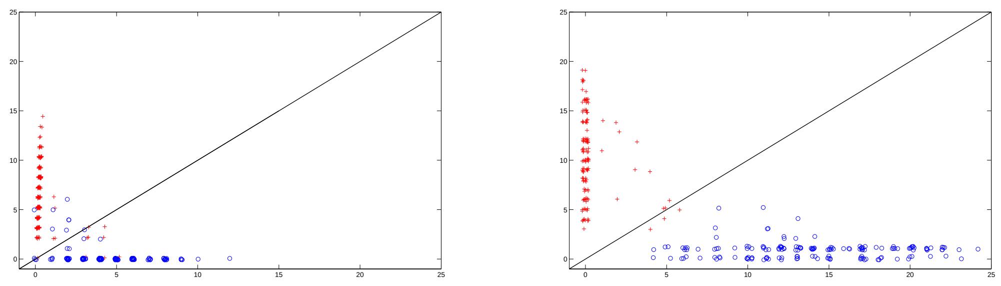

Fig. 1. Output of aRPM algorithm for the Heart data, without and with margin optimization.

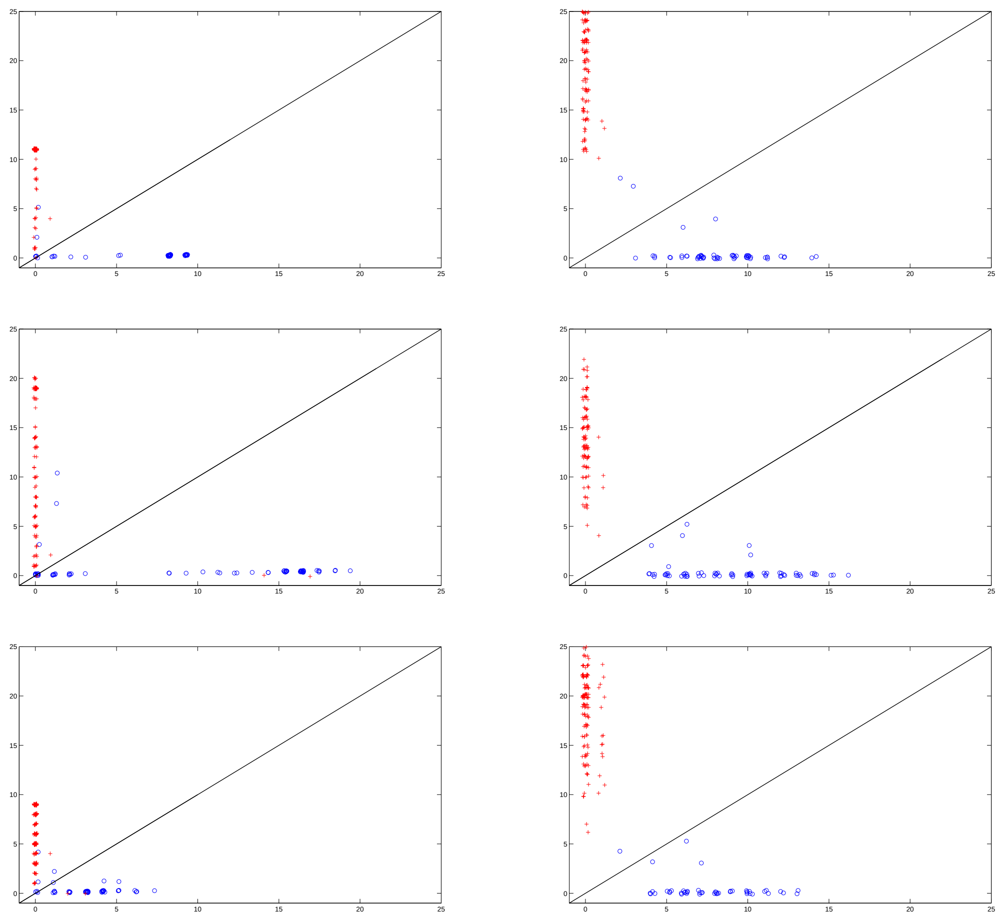

Fig. 2. Output of aRPM algorithm for the Wine data, without (left column) and with (right column) margin optimization. Each row present different class (one class vs rest).

other Boolean functions, not showed here) show that this approach will be particularly useful in solving difficult problems with inherent complex logic. Calculations on much larger datasets from the NIPS 2003 competition [20] are in progress. These datasets have been selected because they are very difficult to analyze correctly by standard MLPs, Support Vector Machines or other machine learning algorithms.

Scatterogram of WTA output shows the effect of margin optimization, and allows for estimation of confidence in classification of a given data. Further improvements to the aRPM algorithm will include admission of impure clusters instead of binary features, with modeling actual $\mathcal{P}(C|t(\boldsymbol{x}))$ distribution along projection directions, use of other kernel features, judicious selection of candidate vectors to define support features and optimization of the algorithm.

References

1. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In D. E. Rumelhart, J.L.M., ed.: Parallel Distributed Processing: Explorations in Microstructure of Congnition. Volume 1: Foundations. MIT Press, Cambridge (1986) 318–362
2. O'Reilly, R., Munakata, Y.: Computational Explorations in Cognitive Neuroscience. MIT-Press (2000)
3. Buonomano, D., Maass, W.: State-dependent computations: Spatiotemporal processing in cortical networks. Nature Reviews Neuroscience **10** (2009) 113–125
4. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation **14** (2002) 2531–2560
5. H. Jaeger, H.H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science **304** (2004) 78–80
6. Jaeger, H., Maass, W., Principe, J.: Introduction to the special issue on echo state networks and liquid state machines. Neural Networks **20** (2007) 287–289
7. Cover, T.M.: Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transactions on Electronic Computers **14** (1965) 326—-334
8. Schölkopf, B., Smola, A.: Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA (2001)
9. Schapire, R., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning **37** (1999) 297—-336
10. Duch, W., Maszczyk, T.: Almost random projection machine. Lecture Notes in Computer Science **5768** (2009) 789–798
11. Duch, W.: k-separability. Lecture Notes in Computer Science **4131** (2006) 188–197
12. Grochowski, M., Duch, W.: Projection Pursuit Constructive Neural Networks Based on Quality of Projected Clusters. Lecture Notes in Computer Science **5164** (2008) 754–762
13. Buesing, L., Maass, W.: A spiking neuron as information bottleneck. Neural Computation **22** (2010) 1–32
14. Brown, D.: N-bit parity networks. Neural Networks **6** (1993) 607–608
15. Grochowski, M., Duch, W.: Learning highly non-separable Boolean functions using Constructive Feedforward Neural Network. Lecture Notes in Computer Science **4668** (2007) 180–189
16. Duch, W., Maszczyk, T.: Universal learning machines. Lecture Notes in Computer Science **5864** (2009) 206–215

17. Grąbczewski, K., Duch, W.: The separability of split value criterion. In: Proceedings of the 5th Conf. on Neural Networks and Soft Computing, Zakopane, Poland, Polish Neural Network Society (2000) 201–208
18. Asuncion, A., Newman, D.: UCI machine learning repository. http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html (2007)
19. Jäkel, F., Schölkopf, B., Wichmann, F.A.: Does cognitive science need kernels? Trends in Cognitive Sciences **13(9)** (2009) 381–388
20. Guyon, I., Gunn, S., Nikravesh, M., Zadeh, L.: Feature extraction, foundations and applications. Physica Verlag, Springer, Berlin, Heidelberg, New York (2006)