

Available from: http://dx.doi.org/10.1109/IJCNN.2011.6033392

Copyright © 2011 IEEE.

This is the author's version of the work. It is posted here with the permission of the publisher for your personal use. No further distribution is permitted. If your library has a subscription to these conference proceedings, you may also be able to access the published version via the library catalogue.

Accessed from Swinburne Research Bank: http://hdl.handle.net/1959.3/208147
Selective Adjustment of Rotationally-Asymmetric Neuron $\sigma$-Widths

Nathan Rose

Abstract—Radial Basis Networks are a reliable and efficient tool for performing classification tasks. In networks that include a Gaussian output transform within the Pattern Layer neurons, the method of setting the $\sigma$-width of the Gaussian curve is critical to obtaining accurate classification. Many existing methods perform poorly in regions of the problem space between examples of differing classes, or when there is overlap between classes in the data set. A method is proposed to produce unique $\sigma$ values for each weight of every neuron, resulting in each neuron having its own Gaussian ‘coverage’ area within problem space. This method achieves better results than the alternatives on data sets with a significant amount of overlap and when the data is unscaled.

I. INTRODUCTION

RADIAL Basis Networks (RBNs) are specialised neural networks utilised for classification tasks. Modeled after statistical classification techniques, they use ‘Pattern’ neurons whose weight values represent an $N$-dimensional point in problem space (where $N$ is the number of input values in the training data examples). The group of weights for a neuron are collectively known as its Exemplar. In contrast to the Perceptron or Back-Propagation neurons, the input values are not multiplied by their respective weights, but rather subtracted. The square-rooted sum of those differences squared thus represents the Euclidean distance between the input data and the neuron’s exemplar. The distance values from multiple trained neurons, combined with a method of evaluating them in the network’s output layer, provide a powerful tool for efficient and parallelisable processing of classification tasks.

Many types of RBN, based on Bayesian classifiers, transform these Euclidean distances via a Gaussian curve to produce the neuron output. The ‘width’ of these curves is controlled by a $\sigma$ parameter and defines the area of problem space within which each neuron has a non-negligible output. There are many methods of determining appropriate values for these $\sigma$’s, but many are rotationally symmetric (i.e., neuron coverage is identical in all directions) and/or non-unique (i.e., all the neurons have the same width). This means that the network may make inaccurate predictions in the region between classes, especially when there is some overlap between classes within problem space.

This paper provides an overview of several established RBN structures and describes their methods of setting the $\sigma$ values. It then proposes a method to produce unique $\sigma$’s for each weight in every neuron. This method is tested against the previous ones and results are compared.

II. RADIAL BASIS NETWORKS

When constructing an RBN, there are many network types from which to choose. This paper does not provide an exhaustive list, but will describe several options in order to clarify the concepts.

One of the simplest network structures is presented by Donald Specht. His Probabilistic Neural Network (PNN) models the Parzen Window statistical method in an easily computable form [1][2]. The weights that make up the exemplars of the neurons in the Pattern Layer are simply copied from the examples in the training set—thus there are as many neurons as there are examples. When presented with a set of inputs, each neuron calculates the Euclidean distance between it and its exemplar as shown in Equation 1 and then applies an output transform, often a Gaussian curve of the form shown in Equation 2. $D_x$ is the Euclidean distance for neuron $x$, $\vec{I}$ is the current input, $N$ is the number of values in the input example, $O_x$ is the output for neuron $x$ and $\sigma$ is the ‘width’ factor.

$$D_x(\vec{I}) = \sqrt{\sum_{j=1}^{N} (X_{xj} - I_j)^2} \quad (1)$$

$$O_x(\vec{I}) = e^{-(\frac{D_x(\vec{I})}{\sigma})^2} \quad (2)$$

The Gaussian output $O_x$ is 1 when the neuron’s exemplar is identical to the input and asymptotes towards 0 as the distance between them increases. The purpose of this is to reduce the effect of the output the further the input is from the neuron, which emphasises the local area of the problem space in classifying the input. The $\sigma$ value controls the ‘width’ of the Gaussian response; the larger it is, the further away from the neuron an input can be and still produce a non-negligible output. There are many methods for setting the $\sigma$ value, several of which will be described in the next section.

The second layer of the PNN, known as the Summation Layer, has one neuron per class in the input data. As the neurons in the Pattern Layer are trained, their output is connected to the Summation neuron that represents their respective training example’s known class. When an input is presented to the network, these neurons sum the Pattern Layer outputs to produce a Probability Density Function (PDF) that represents the probability that the new input is a member of each class. When there are uneven numbers of examples (and thus neurons) of each class in the training data, some versions of the PNN divide the PDF by the number of neurons to produce an average value. The PDFs are passed to the final layer, which simply chooses the highest one and sets the output to that class’s ID.
Disadvantages of the PNN include the amount of memory needed to store every training example as a neuron and the processing required to calculate the distance and output for every one. Moody and Darken [3][4] suggest merging the training examples using a k-means clustering technique. This groups the training examples in a pre-set number of clusters, the centres of which becomes the locations of the neurons. The outputs from the Pattern Layer are fed to an ADALINE or feed-forward network. After the pattern neuron’s $\sigma$’s have been set, the third stage of training is a supervised process—a training example is presented to the Pattern Layer, the neuron outputs are fed to the output layer and its response is trained towards the example’s class ID. This is repeated for all the examples until the ADALINE (or network) is trained. Predicting the class of new examples follows a similar path: the example is input to the Pattern Layer, the Pattern neurons’ outputs are input to the ADALINE and the ADALINE’s output is the prediction.

Verleysen and Hlaváčková [5] replace the k-means clustering in Moody and Darken’s RBN (MD-RBN) with a competitive learning approach that produces similar results while also being adaptive in real time. Each example in the training set is presented to the network one at a time and the closest neuron is found. If the Euclidean distance between the example and the neuron’s exemplar is larger than a user-chosen Difference Threshold, a new neuron is created. If not, the weights in the neuron’s exemplar are trained towards the new example as shown in Equation 3, where $l$ is the learning rate. A high threshold will result in fewer neurons being created and thus less accurate classification, whereas a very low threshold will produce a Pattern Layer identical to that which would have been produced by the PNN. Unlike the PNN, this training is performed in an unsupervised fashion—examples from differing classes may be merged together—so a more complicated output layer is needed to predict the class.

$$X_{x_{j_{new}}} = (1 - l) \times X_{x_{j_{old}}} + l \times I_3$$ (3)

Kohonen’s LVQ network [6] has several variations and the training method differs subtly between them. The primary structural differences, in comparison with the PNN and the MD-RBN, are as follows. Neurons in the Pattern Layer are pre-assigned to classes. Similar to Verleysen and Hlaváčková’s training method, new examples are merged with the existing exemplars, but only within the same class (though there are several more training rules beyond that). Also, when a new example is presented to the network, the predicted class is simply that of the closest neuron to the example.

III. METHODS FOR SETTING $\sigma$

An adequate value for $\sigma$ is as important as finding the right neuron exemplars with respect to the classification accuracy of the network. The $\sigma$ value determines the width of the Gaussian transform and thus the Voronoi regions, which are the regions of problem space within which a given neuron is closer than any other. An ideal $\sigma$ will result in overlap of the coverage areas for the neuron within each class, but a clear delineation between the classes. If $\sigma$ is too small, the network will not generalise about areas of the problem space close to the known training examples, however if $\sigma$ is too large, overlap between the classes may result in inaccurate predictions.

Specht does not specify a specific approach to determine an appropriate $\sigma$ value; he suggests experimenting with multiple values and using the one that results in the lowest error [1][7]. Moody and Darken [3] suggest three different methods. The first method is to minimise the value of Equation 4 with respect to $\sigma^\alpha$, where $P$ is an overlap parameter and $n$ is the number of neurons. This calculates the effective Euclidean distance between the neuron exemplars and ensures that the Gaussian widths do not overlap more than $P$—note that this formula does not present any inputs to the neurons, rather each neuron calculates the distance between its own exemplar ($\vec{X}_\alpha$) and each of the others ($\vec{X}_\beta$). It should be also noted that Moody and Darken here allow each neuron to have its own unique $\sigma$ value to reflect the distance between itself and the closest neurons.

$$E = \frac{1}{2} \sum_{\alpha=1}^{n} \left[ \sum_{\beta=1}^{n} O_x(\vec{X}_\beta) \left( \frac{\vec{X}_\alpha - \vec{X}_\beta}{\sigma^\alpha} \right) - P \right]$$ (4)

However, Moody and Darken also provide two heuristic approaches that provide “qualitatively similar results” [3]. The first, the ‘P Nearest Neighbours’ heuristic, sets $\sigma^\alpha$ for each neuron to the root mean square value of the Euclidean distances to the $P$ nearest neurons. The second, the ‘Global First Nearest Neighbours’ heuristic, applies the P Nearest Neighbours approach with $P = 1$ and sets the $\sigma$ of every neuron to the average value of $\sigma^\alpha$. Moody and Darken actually recommend using the latter approach and the results below show that it regularly results in better prediction accuracy than P Nearest Neighbours.

It should be noted that these methods only consider the distribution of the neurons within problem space and not the desired output. Both Masters [2] and Guillén et al [8] recommend approaches that incorporate the training data in the calculation of $\sigma$-widths.

Masters’ approach involves multiple $\sigma$ values, one for each input weight but shared identically by all the Pattern neurons. Equations 1 and 2 are now changed to Equations 5 and 6, respectively. This means that $D_x$ is now not strictly a Euclidean distance, but a weighted distance metric. While the $\sigma_j$ values are not unique between neurons, this results in the neuron Gaussian output being different in every direction; rotationally asymmetric.

$$D_x(\vec{I}) = \sqrt{\sum_{j=1}^{N} \left( \frac{X_{x_j} - I_j}{\sigma_j} \right)^2}$$ (5)

$$O_x(\vec{I}) = e^{-(D_x(\vec{I}))^2}$$ (6)
This has advantages in terms of handling unevenly spread input data. $\sigma_j$ can now be regarded as a scaling factor, so the training data set need not be scaled in advance. While scaling is normally a simple procedure, it can be difficult on large data sets and is impossible in real-time, dynamic systems.

If calculated appropriately, the $\sigma_j$ values can also be observed as an indicator of the ‘importance’ of each input in the problem space. Some of the data inputs may have a stronger correlation with the examples’ classes than others; it may be found that these inputs have significantly larger $\sigma_j$ values, thus emphasising their effect on the overall neuron output. By observing the $\sigma_j$ values after the training process is complete, qualitative insight on the training set can sometimes be derived.

IV. LIMITATIONS TO EXISTING APPROACHES

Given that Masters recommends giving each input its own $\sigma_j$, and Moody and Darken suggest (in two of their three approaches) giving each neuron a unique $\sigma$, the logical next step would be to give each neuron its own unique $\sigma$ value. The distance metric for this type of neuron is shown in Equation 7 and the neuron output will be the same as Equation 6. This should result in more accurate predictions in the boundary regions between classes (in problem space) and less need to scale data. It is also a recognition of the fact that problem spaces are often not made up of homogeneous, well-spaced sets of examples but are often irregular and asymmetric.

$$D_x(I) = \sqrt{\sum_{j=1}^{N} \left( \frac{X_{xj} - I_j}{\sigma_{xj}} \right)^2} \quad (7)$$

For example, in the Iris data set [9] the Iris Setosa class is always linearly separable from the other classes. When observing each input measurement individually, it is clear that there is strong ‘overlap’ between all three classes in the Sepal Length (SL) and Sepal Width (SW) measurements. However, the Iris Setosa examples are cleanly separated in the Petal Length (PL) and Petal Width (PW) measurements. This implies that this class’s data can be stored in a small number of neurons with small $\sigma_j$ values for the Sepal measurements and large values for the Petal measurements. Iris Versicolor and Iris Virginica experience overlap in all four input measurements, but have a larger number of cleanly-separable examples in the two Petal measurements.

Figure 1 shows the distribution of data points for two of the measurements: Sepal Length and Petal Width. As described, all three classes overlap strongly on the SL-axis. Iris Setosa is easily separable in the PW-axis and the other two classes only show a small amount of overlap.

These classes’ neurons would likely have large $\sigma_j$ values for these measurements within the ‘clean’ regions but smaller in the ‘overlap’ region, in order to finely delineate the separation between the classes. A ‘One-$\sigma$-Fits-All’ approach on this data set would either perform inaccurately in the ‘border’ regions, or would require the same (higher) density of neurons in the ‘clean’ regions as in the ‘border’, which would be a waste of memory and processing power.

V. SELECTIVE ADJUSTMENT METHOD

A general description of a $\sigma$-setting method that can be applied to any Radial Basis Network with only small modifications is given first. It is followed by a more detailed discussion of the method as implemented in this work.

The Pattern Layer of the RBN is trained via any method. The $\sigma$ values within the Gaussian transform are all initialised to the same non-zero value. For the purposes of testing, a starting value of 1 was used in order to obtain significantly different $\sigma$ values in comparison to the other methods, although it may be beneficial to initially set the $\sigma$’s via one of Moody and Darken’s heuristic methods and then use this method to refine them further. $\sigma$’s of 1 result in the ‘distance’ metric being exactly equivalent to Euclidean distance, which may be convenient in the initial stages of training.

The entire training dataset is presented to the network, one example at a time. If the network correctly predicts the example’s class, no further adjustment is needed. However, if the class is incorrectly predicted, the $\sigma$ values in multiple neurons will be adjusted. Depending on the type of output layer used, it may be difficult to ascertain which neurons are ‘responsible’ for the incorrect prediction. While it may seem logical to update the $\sigma$’s in all the neurons, this would be both computationally expensive and, ultimately, futile as it would always result in all of the neurons sharing the same values. The chosen heuristic is to find the closest neuron with the same class ID as the example being tested (this precludes a strict implementation of the MD-RBN, as it requires the neurons to be trained in an unsupervised fashion), as well as all the other neurons (with incorrect class IDs) located ‘in between’ this neuron and the example. Rather than directly alter the $\sigma$ values stored in each neuron, a vector of coefficients to the $\sigma$’s, C, is created with all values initially 1. This vector represents the change to be made to the $\sigma$’s for

![Fig. 1. Sepal Length and Petal Width in the Iris data set](image-url)
the relevant neurons. Thus, these neurons all experience the same change in $\sigma$, however as multiple changes are made to different subsets of neurons, their actual values will diverge. The neurons now calculate their ‘distance’ to the example using Equation 8.

$$D_x(\vec{I}) = \left[ \sum_{j=1}^{N} \left( \frac{X_{xj} - I_j}{\sigma_{xj} \times C_j} \right)^2 \right]$$  \hspace{1cm} (8)

The method of calculating the error in the output of the network will depend on the network structure used. One method derived by Masters [2], drawing from Schiøler and Hartmann [10], calculates error for a PNN-style output layer. The Probability Density Function $h_k(\vec{I})$ of each class is the sum of the neuron outputs for each class as shown in Equation 9, where $k$ is the class ID, $x$ is the index of the neurons, and $\delta_k$ is a function that equals 1 if neuron $x$ is a member of class $k$ and 0 otherwise.

$$h_k(\vec{I}) = \sum_{(x=1)}^{n} \delta_k(x) O_x(\vec{I})$$  \hspace{1cm} (9)

The Bayesian confidences ($b_k$ in Equation 10, where $k$ is the class ID, and $K$ in Equation 11 is the number of classes) for each class are the PDF for that class divided by the total PDF for all classes. This makes intuitive sense; if the network is classifying accurately, the PDF for the correct class will be high and the PDF for all other classes will be 0, leading to a $b_k$ of 1.

$$b_k(\vec{I}) = \frac{h_k(\vec{I})}{s(\vec{I})}$$  \hspace{1cm} (10)

$$s(\vec{I}) = \sum_{k=1}^{K} h_k(\vec{I})$$  \hspace{1cm} (11)

The error in the Bayesian confidences is shown in Equation 12, which represents the error for the given input ($\vec{I}$) over the subset of neurons ‘responsible’ for the mistaken class prediction. This error must be minimised with respect to the $C$ $\sigma$-coefficients.

$$e_k(\vec{I}) = \left[ 1 - b_k(\vec{I}) \right]^2 + \sum_{j \neq k} \left[ b_j(\vec{I}) \right]^2$$  \hspace{1cm} (12)

Any minimization technique may be used, though an algorithm which does not evaluate the error many times will have an advantage as this evaluation will be computationally expensive when the number of neurons is large. A heuristic approach is ideal; the global best $C$ is not necessary as the $\sigma$’s in each of the neurons will likely change multiple times during the overall adjustment process for the network. The test results below utilised the Conjugate Gradient minimization technique to minimise $C$, with Golden Section line-minimization.

When an acceptable value for $C$ is found, the changes are made permanent in the set of ‘responsible’ neurons by adjusting all their $\sigma$ values using Equation 13.

$$\sigma_{xjnue} = \sigma_{xjold} \times C_j$$  \hspace{1cm} (13)

This process is performed for all the training examples and repeated until the network predicts the class correctly for every one. This, however, may be impossible and it is advisable to set an upper limit on the number of iterations that the adjustment process can repeat before abandoning its search.

VI. TEST PROCEDURE

The proposed method of setting $\sigma$ values can, in theory, be used with any network structure that utilises a Gaussian output transform. A test harness was developed that implemented several different $\sigma$-setting techniques and output layers in order to test and compare them in a consistent manner.

A series of neurons were initialised and trained using the method suggested by Verleysen and Hlaváčková, with one significant difference: training was not an unsupervised process. This was necessary as the process of calculating error required definite ‘correct’ and ‘incorrect’ responses. To implement this, each neuron was assigned a class ID number matching the class of the first example with which it was trained.

Describing this method in detail, the examples in the training dataset were presented to the neurons one at a time. Neurons whose class ID matched that of the training example all calculated the Euclidean distances from their exemplars to the current example’s location in problem space. If the closest neuron was further than a user-set Difference Threshold (or if no neurons of that class existed), a new neuron was initialised that simply copied the example into its exemplar and set the class ID to match. Conversely, if the closest neuron’s location was within the Difference Threshold, its exemplar was merged with the new example using Equation 3. In this fashion, the Pattern Layer was trained with all of the examples in the training dataset.

Four of the previously described methods of setting the $\sigma$ values were implemented:

- Setting all the $\sigma$’s to the same value, in this case $\sigma = 1$. (cf. the PNN).
- Setting the $\sigma$’s for each neuron separately, using Moody and Darken’s ‘P Nearest Neighbours’ (P-NeNe) heuristic.
- Setting all the $\sigma$’s to the same value using Moody and Darken’s ‘Global First Nearest Neighbour’ heuristic (G-FNeNe).
- Setting all the $\sigma$’s to the same value (1, by default) and then selectively adjusting them when testing on the training set produced errors, using the described selection procedure and Conjugate Gradient minimisation—‘Selective Adjustment’ (SA).

Once the neuron centres in the Pattern Layer were trained, each of these methods were applied one at a time (after resetting the $\sigma$ values to 1). It must be emphasised that these methods only alter the $\sigma$ values within the Gaussian output
transform and do not change the exemplar values within the
neurons. Each method was thus applied to an identical set
of neurons and tested as such. Three different output layers
were also implemented and were each tested with all of the
σ values, giving 12 sets of results for each trained network.

- ‘Nearest Wins’ (NW) – the stored class of the neuron
with highest output is chosen as the predicted class of
the test item.
- ‘Probability Density Function’ – cf. the PNN, the output
of the neurons for each class is summed and averaged
to produce a PDF and the class with the highest is chosen.
- ‘ADALINE’ – cf. the MD-RBN, each neuron’s output
was passed as input to an ADALINE element that was
trained in a supervised fashion on the Pattern Layer’s
response to all of the examples in training data.

The networks were trained and tested using the ‘Round
Robin’ method. The examples in the training data were
randomly assigned to one of ten subsets. Ten networks with
identical setup parameters were each trained on 90% of the
data and tested on the remaining 10%. The number of errors
from each of these networks were summed to produce the
overall error level for that network setup. This process was
then repeated 50 times and the results averaged. The number
of neurons trained in each network’s Pattern layer was also
recorded and averaged—this will generally decrease as the
Difference Threshold increases, but can even vary for the
same Difference Threshold due to the randomised order of
training examples.

In order to obtain a wide set of results, the networks
were trained with two data sets in both scaled and unscaled
(original data) form. These were obtained from the UCI
Machine Learning Repository [11] and were chosen as they
each have significantly different numbers of inputs, classes
and range of values in the unscaled form. The data sets are
as follows:

- Iris data set: 150 examples, 4 inputs, 3 output classes
[9].
- Glass Identification data set: 214 examples, 9 inputs, 6
output classes [12].

Each of these sets was scaled to a range of 0–1 (by
column). As the number and positioning of the neurons in the
Pattern Layer affects the accuracy of all the network types,
a range of Difference Thresholds was tested for each of the
data sets. The scaled sets were tested with thresholds from
0.01–1.0 in steps of 0.03 and the unscaled sets were tested
as follows:

- Fisher’s Iris data set: 0.01–5.81 in steps of 0.2.
- Glass Identification data set: 0.01–1.0 in steps of 0.03
and 1.3–10.0 in steps of 0.3.

VII. RESULTS

A. Iris Data Set

Figure 2 shows the average number of errors (i.e., misclas-
sified examples) for the Scaled Iris data set for the NW, PDF
and ADALINE outputs. The x-axis of the plots is the average
number of neurons in the Pattern Layer of the network for
a given Difference Threshold. As expected, most of the σ-
setting methods and output layers show fewer errors for
larger number of neurons, as this allows the network to hold a
more complex model of the problem space. This is especially
true for the Selective Adjustment method, as larger numbers
of neurons allow more finely-detailed adjustment of the σ’s.

In the figures that present the NW outputs, the PNN lines
are not visible as their results are identical to the G-FNeNe.
This is because both these methods produce identical σ’s
for all the weights in all the neurons, meaning that all the
neurons are scaled equally in the Gaussian and the ‘closest’
neuron is unaffected. The other two outputs produce different
results for these methods as they combine the output from
multiple neurons, so the size of the neuron’s active area in
problem space will be relevant.

It can be seen that the G-FNeNe method performs well
across all output types and for a wide range of Pattern Layer
sizes. The SA method performs poorly when there are a small
number of neurons but better as the Pattern Layer increases in
size. With the NW output (Figure 2(a)), SA performs better
than G-FNeNe when the number of neurons is higher than
approximately 15. However, this average result does not take
in to account the reliability of the network. The three prior
methods produce a fairly consistent number of errors across
a long test, whereas SA tends to either produce a very low
or very high number of errors. For instance, at a Difference
Threshold of 0.08, the Pattern Layer had an average (across
50 runs) of 16.85 neurons, G-FNeNe had an average error
of 7.04 and SA had an average error of 7.28. However, despite
having an overall average higher error, SA resulted in a lower
number of misclassifications than G-FNeNe in 29/50 runs;
higher number in 17/50 runs and equal in 4/50.

SA performed poorly compared to the other networks
with both the PDF and ADALINE outputs. The latter is
unsurprising, as this is the output type with which Moody
and Darken developed their heuristic methods. The ADALINE
results were similar for all the other data sets, and have not
been shown due to space restrictions. The fact that SA in
general performs well with the NW output is probably due
to the fact that the selection of neurons to adjust is also via
the Nearest-Wins approach.

The results for the Unscaled Iris data, shown in Figure 3
show similar results for the NW output but a marked im-
provement for the SA method with the PDF method. It is
likely that this is because of SA’s flexibility when dealing
with unscaled data. However, while its error comes close to
that of the G-FNeNe method, it is still less accurate over the
entire range.

B. Glass Data Set

Figure 4 shows the results for the Scaled Glass data
set. With the NW output, the SA method performs slightly
worse than G-FNeNe at low numbers of neurons, though
comparatively better than it did on the Scaled Iris set. At
larger numbers of neurons, SA performs significantly better
than the alternatives. With the PDF output, SA performed
better than the other methods for the entire range of tests.
This is not surprising, as the calculation of $e_k$ (Equation 12) in this method uses the PDF output.

The results for the Unscaled Glass data set in Figure 5 show that the SA method performs significantly better on the unscaled data than the scaled. The NW output is similar to those of the previous data sets—SA performs better when there are more neurons. The PDF output shows SA being more sensitive to the number of neurons than in the scaled data. This is probably due to the range of values in each input of this data set being so diverse—scaling smooths this out, allowing better results at smaller numbers of neurons. However, at larger numbers of neurons, SA’s flexibility allows more accuracy.

VIII. Discussion

The results demonstrate that while Selective Adjustment is not a perfect method of setting neurons’ $\sigma$ values for every problem, it provides good results in many contexts and significantly better results than the alternatives on some data sets. It is highly sensitive to the number of neurons in the network’s Pattern Layer and performs poorly when the
Fig. 4. Results for Glass Scaled data

(a) NW Output

(b) PDF Output

Fig. 5. Results for Glass Unscaled data

number of neurons is low. This is likely because SA produces a more flexible and complicated map of the class locations in problem space, which can be more accurate with higher numbers of neurons. In comparison, the Global-First Nearest Neighbours method has the same sized Gaussian ‘window’ for all the neurons; if there are fewer neurons, the windows will be bigger, but overall will still cover approximately the same regions.

SA excels in the finer regions between classes. This is evident in the difference between the Iris data results and the Glass data results. The Iris data contains three classes, one of which is located as a clean, separate group in problem space. The other two classes only have a small amount of overlap. On the Iris set, all the methods produce good results but SA is outmatched by its competition. On the other hand, the Glass data set contains six classes and a very large amount of overlap. The other three methods do not do a good job of separating the classes with the Gaussian windows. SA, however, performs much better and produces much more accurate predictions, particularly with the PDF output layer. With both data sets, SA produced fewer errors compared to the alternatives when tested on the unscaled data than on the scaled data, which demonstrates its flexibility in cases where scaling may be impossible, for instance large data sets or real-time data.

It should be possible to improve the results produced by the Selective Adjustment method by making small adjustments to the software as tested. During Pattern Layer training, each example in the training set must be within the Difference Threshold of one of the neuron’s exemplar. However, as successive examples are shown, that exemplar may ‘move’ away from its original example. This could be rectified during the $\sigma$-setting stage by checking whether the closest neuron of the correct class is within the Difference Threshold and adding a new neuron when it is not.

As described in the previous section, when SA performs poorly, it tends to produce very bad results. It should be possible to detect when this happens by observing the number of misclassifications at the end of the process. When this is beyond a user-chosen ‘acceptable’ level, the process could repeat by resetting all the $\sigma$’s back to 1 and re-randomizing the order of the training examples. It is likely that presenting
the examples in a different order will produce different $\sigma$ values, which may produce better results.

Given SA’s focus on the edges between class regions in problem space, it may be possible to remove neurons from within large clumps of neurons with the same class ID. The few neurons left within the group would be able to increase their $\sigma$-widths and thus represent the entire inner region. This would mean that with a data set like the Iris data, SA would be able to achieve similar (if not identical) accuracy with fewer neurons and thus may produce better results than the Global-First Nearest Neighbours method.

There will never be one method of determining $\sigma$ values for the Pattern Layer neurons that will produce perfect results for all data sets. There are a large number of techniques, including many not examined in this paper. Moody and Darken’s Global-First Nearest Neighbour is a good all-purpose method that produced reasonably good results on all of the data sets tested. However, for unscaled data and sets that have large amounts of class overlap, the Selective Adjustment method will, in many cases, provide better results and clearer delination between classes.

REFERENCES