Information Sciences 178 (2008) 3802–3812

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A modal learning adaptive function neural network applied to handwritten digit recognition

Miao Kang, Dominic Palmer-Brown *

School of Computing and Technology, University of East London, 4-6 University Way, London E16 2RD, United Kingdom

ARTICLE INFO

Article history:
Received 2 November 2007
Received in revised form 7 March 2008
Accepted 13 May 2008

Keywords:
Modal learning
Adaptive function neural network
Snap–drift neural network
Handwritten digits recognition
Piecewise linear function

ABSTRACT

A novel combination of the adaptive function neural network (ADFUNN) and on-line snap–drift learning is presented in this paper and applied to optical and pen-based recognition of handwritten digits [E. Alpaydin, F. Alimoglu for Optical Recognition of Handwritten Digits and E. Alpaydin, C. Kaynak for Pen-Based Recognition of Handwritten Digits http://www.ics.uci.edu/~mlearn/databases/optdigits/ http://www.ics.uci.edu/~mlearn/databases/pendigits/]. Snap–drift [S.W. Lee, D. Palmer-Brown, C.M. Roadknight, Performance-guided neural network for rapidly self-organising active network management (Invited Paper), Journal of Neurocomputing, 61C, 2004, pp. 5–20] employs the complementary concepts of common (intersection) feature learning (called snap) and LVQ (drift towards the input patterns) learning, and is a fast, unsupervised method suitable for on-line learning and non-stationary environments where new patterns are continually introduced. ADFUNN [M. Kang, D. Palmer-Brown, An adaptive function neural network (ADFUNN) for phrase recognition, in: The International Joint Conference on Neural Networks (IJCNN05), Montréal, Canada, 2005, D. Palmer-Brown, M. Kang, ADFUNN: An adaptive function neural network, in: The 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA05), Coimbra, Portugal, 2005] is based on a linear piecewise neuron activation function that is modified by a novel gradient descent supervised learning algorithm. It has recently been applied to the Iris dataset, and a natural language phrase recognition problem, exhibiting impressive generalisation classification ability with no hidden neurons. The unsupervised single layer snap–drift is effective in extracting distinct features from the complex cursive-letter datasets, and the supervised single layer ADFUNN is capable of solving linearly inseparable problems rapidly. In combination within one network (SADFUNN), these two methods are more powerful and yet simpler than MLPs, at least on this problem domain. We experiment on SADFUNN with two handwritten digits datasets problems from the UCI Machine Learning repository. The problems are learned rapidly and higher generalisation results are achieved than with a MLP.

1. Introduction

Twenty years ago there were already several forms of artificial neural network, each utilising a different form of learning. At that time it was considered likely that a very small number of forms of learning would prevail and become ubiquitous. Along the way, many types of learning, notably back-propagation (BP), Bayesian and Kernel methods have been hailed as superior forms. Long after their introduction, Kohonen learning, self-organising maps (SOMs), and back-propagation are still being used alongside more recent methods such as Bayesian [16] and SVMs [4]. A wide range of methods are in use, simply

* Corresponding author. Tel.: +44 (0) 20 8223 2170.
E-mail address: D.Palmer-Brown@uel.ac.uk (D. Palmer-Brown).

0020-0255/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2008.05.011

because they work. There are significant problems and datasets for which each method is suitable and effective. Within this context, the modal learning concept arises from the desire to equip a single neural network or module with the power of several modes of learning, to achieve results that no single mode of learning can achieve.

In this paper we combine two modal learning neural networks, snap–drift [9–11] and ADFUNN [7,12] on a complex handwritten cursive-letter recognition task. The original motivation of snap–drift was to improve performance by swapping to or staying with the mode which gives better performance, e.g. reinforcing the mode that works. Toggling the mode continuously is also invaluable as a means of purely unsupervised learning and has been used in natural language processing with modal adaptation at the neuron level, improving on the performance of MLPs with BP.

In ADFUNN (an Adaptive function neural network), two-mode adaptation is applied simultaneously between and within neurons. The conventional neurocomputing wisdom is that by adapting the pattern of connections between neurons the network can learn to respond differentially to classes of incoming patterns. The success of ANNs in an age of massively increasing computing power that has made high speed neurocomputing feasible on the desktop and more recently in the palm of the hand, has resulted in little attention being paid to the implications of adaptation within the individual neurons. The computational assumption has tended to be that the internal neural mechanism is fixed. However, there are good computational and biological reasons for examining the internal neural mechanisms of learning. Recent neuroscience suggests that neuromodulators play a role in learning by modifying the neuron's activation function [14,15] and with an adaptive function approach it is possible to learn linearly inseparable problems fast, even without hidden nodes.

The snap–drift learning algorithm was first introduced by Palmer-Brown and Lee [9–11] and it emerged as an attempt to simplify and modify Adaptive Resonance Theory (ART) learning in non-stationary environments where self-organisation needs to take account of periodic or occasional performance feedback [9,10]. Since then, the snap–drift algorithm has proved invaluable for continuous learning in several applications. In a snap–drift network, snap is based on logical intersection, which is implemented as a fuzzy AND; and drift is based on learning vector quantization (LVQ) [8]. Snap–drift harnesses the complementary strengths of the two modes of learning which are dynamically combined in a rapid form of adaptation.

The unsupervised single layer snap–drift is very effective in extracting distinct features from the complex cursive-letter datasets, and it helps the supervised single layer ADFUNN to solve these linearly inseparable problems rapidly without any hidden neurons. Experimental results show that in combination within one network (SADFUNN), these two methods are more powerful and yet simpler than MLPs.

2. Modal learning neurons

With the combination power of several modes of learning, a modal learning neural network can achieve results beyond the scope of any single mode, through exploitation of the complementary nature of each mode. A mode is defined as an adaptation method for learning that could be applied in more than one type of architecture or network. It is analogous to a human mode of learning, such as rote learning, learning by analogy or category learning. Modes of learning map onto neural network learning objectives. Hence, well known example of modes include the Delta Rule, BP, learning vector quantization (LVQ) and Hebbian Learning; but not Adaptive Resonance Theory (ART) or Bayesian Neural Networks, since those are architectures and approaches that define more than a mode of learning.

An important feature for modal learning is that the actual objective of learning may be unknown, changing or difficult to quantify. Even if a learning objective is easy to define in terms of a desired learning outcome (e.g. zero or minimal error), the learning agent's objective function is different because it must be expressed in terms of the parameters of the learning machine (e.g. de/dw). Given the dislocation between the objectives of learning and the objective learning function of the learning agent, it is not possible to know a priori which is the optimal learning mode to use. For example, BP is good for approximation and transformation but if features within the data need to be assimilated directly, then learning vector quantization or SVM may be more effective. We are unlikely to know in advance, but the learning agent should be able to determine the optional learning mode during, and as part of, the learning process.

Modal Learning contrasts with hybrid and modular approaches in which each module or network exhibits one mode, and modules are bolted together with each designed to solve a sub-problem. In general, in a hybrid approach, increasing the number of modes leads to increasing the number of neurons, layers or networks. Yet, even a fixed learning task and data set may benefit from sequential or simultaneous application of more than one mode, and non-stationing data is even more likely to benefit from or necessitate mode changes.

3. Related work on methods of handwritten cursive-letter recognition

Handwritten letter recognition task has a long history and always been a challenging problem encountered in many real-world applications, such as postal mail sorting, bank check recognition, and automatic data entry from business forms. A good computational solution must have the ability to recognize complex cursive-letter patterns and represent commonsense knowledge of them.

Handwritten cursive-letter recognition problem is made complex by the fact that the writing is fundamentally ambiguous as the strokes in the letter are generally linked together. The recognition techniques can be classified according to two criteria: the way preprocessing is performed on the data and the type of the decision algorithm.

Artificial neural networks can finely approximate complicated decision boundaries. It is one of the most popular methods which applied to this area with the first example dating back to 1950s. Letter recognition was included in Frank Rosenblatt's [13] perceptron neurocomputer. It was one of the first computers based on the idea of a neural network, which is a simplified computational model of neurons in a human brain. It was the first functioning neurocomputer, and it was able to recognise a fixed-font character set. A feed forward neural network trained by Quickprop algorithm, which is a variation of error back propagation is used for on-line recognition of handwritten alphanumeric characters by Chakraborty [5]. Some distorted samples in numerals 0–9 are used as experiment which are different from the dataset used in this paper. Good generalization capability of the extracted feature set is reported.

Along the way, many neural network learning methods like (Multi Layer Perceptron)MLP, k Nearest Neighbours (kNNs), RBF etc. have been applied to this area but the difficulty of handwriting recognition has always been underestimated.

Zhang and Li [17] propose an adaptive nonlinear auto-associative modelling (ANAM) based on locally linear embedding (LLE) for learning the cursive optical and pen-based letter recognition tasks [3] which we use in this paper from UCI repository. LLE algorithm is a modified k-NN to preserve local neighbourhood relation of data in both the embedded Euclidean space and the intrinsic one. In ANAMs, training samples are projected into the corresponding subspaces. Based on the evaluation of recognition criteria on a validation set, the parameters of inverse mapping matrices of each ANAM are adaptively obtained. And then that of the forward mapping matrices are calculated based on a similar framework. 1.28% and 4.26% error rates can be obtained by ANAM for optical recognition and pen-based recognition respectively. However, given its complex calculation of forward mapping and inverse mapping matrices, many subspaces are needed and also suboptimal auto-associate models need to be generated. SADFUNN is computationally much more efficient, simpler and achieves similar results. It will be a straight forward process to apply it to many other domains.

4. A single layer adaptive function network (ADFUNN)

ADFUNN provides a means of solving linearly inseparable problems using a simple adaptive function neural network (ADFUNN), based on a single layer of linear piecewise function neurons, as shown in Fig. 1.

ADFUNN is composed by a layer of input neurons which fully forward connected to a layer of output neurons. Like the classic neural network structure, the nodes of the input layer are passive, meaning they do not modify the data. They receive a single value on their input, and duplicate the value to their multiple outputs. In comparison, the nodes of the output layer are active. This means they modify the data. The values entering an output node are multiplied by weights, a set of predetermined numbers stored in the program. The weighted inputs are then added to produce a single number. Before leaving the node, this number is passed through a piecewise linear activation function. This is continuous piecewise curve that limits the node's output. That is, the input to the piecewise function is a value within a given range and its output can only be between 0 and 1. ADFUNN is introduced to overcome the linear inseparability limitation in a single weight layer supervised network. In ADFUNN, weights are adapted using an improved delta learning rule. The delta learning rule, change in weight from u_i to u_j is given by:

$$\Delta w_{ij} = r \cdot a_i \cdot e_j,$$

where r is the learning rate, a_i is the activation of u_i and e_j is the error from output u_j. From considerations of simple calculus, multiplying by the slope ensures the correct adaptation direction from a point on the function curve, and achieves more accurate and faster learning. Therefore, in ADFUNN, the delta learning rule is modified as:

$$\Delta w_{ij} = r \cdot a_i \cdot e_j \cdot s_j,$$

where s_j is the slope of the line which runs through two neighbouring f-points. We calculate $\sum aw$, and find the two neighbouring f-points that bound $\sum aw$. Two proximal f-points are adapted separately, on a proximal-proportional basis. The

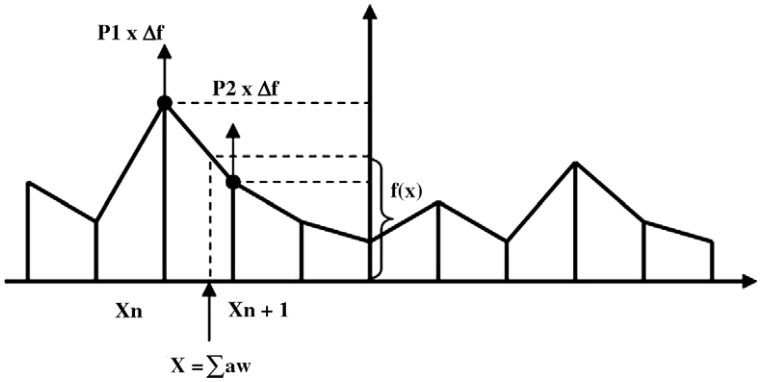

Fig. 1. Adapting the linear piecewise neuronal activation function in ADFUNN.

proximal-proportional value $P1$ is $(Xn+1-x)/(Xn+1-Xn)$ and value $P2$ is $(x-Xn)/(Xn+1-Xn)$. Thus, the change to each point will be in proportion to its proximity to x. We obtain the output error and adapt the two proximal f-points separately, using a function modifying version of the delta rule, as outlined in Section 4.1 to calculate Δf.

4.1. The general learning rule for ADFUNN

The weights and activation functions are adapted in parallel, using the following algorithm:
A = input node activation, E = output node error.
WL, FL: learning rates for weights and functions.

Step 1: Calculate output error, E, for input, A.
Step 2: Adapt weights to each output neuron:
$\Delta w = WL \cdot Fslope \cdot A \cdot E$
$w' = w + \Delta w$
weights normalisation
Step 3: Adapt function for each output neuron:
$\Delta f(\sum aw) = \mathrm{FL} \cdot \mathrm{E}$
$f_1' = f_1 + \Delta f \cdot P1, f_2' = f_2 + \Delta f \cdot P2$
Step 4: $f(\sum aw) = f\prime(\sum aw)$;
$w = w'$.
Step 5: Randomly select a pattern to train
Step 6: Repeat steps 1–5 until the output error tends to a steady state.

4.2. XOR Experiment using ADFUNN

The XOR is a simple binary example of linear inseparability, and therefore serves as a good basic test to establish that linearly inseparable problems can be solved by ADFUNN. Two weights are needed for the two inputs and there is one output. Weights are initialized randomly between -1 and 1, they are then normalised. F-point values are initialized to a constant value of 0.5. Each F-point is simply the value of the activation function for a given input sum. F-points are equally spaced with an interval of 0.4, and the function value between points is on the straight line between them. This network is adapted using the above general learning rule.

The ADFUNN learns the problem very fast with a learning rate of 0.5. An example of the weights after learning is: $w_1 = 0.62$, $w_2 = 0.73$, and therefore, the sum of weighted inputs $w_1^*0 + w_2^*0 = 0$ for input pattern (0, 0), $w_1^*0 + w_2^*1 = 0.73$ for input pattern (0, 1), $w_1^*1 + w_2^*0 = 0.62$ for input pattern (1, 0) and $w_1^*1 + w_2^*1 = 1.35$ for input pattern (1, 1).

As can be seen in Fig. 2, a characteristic XOR curve is learned. The raised curve (between 0 and 1.2) marks the learned region, within which adaptation has occurred. The data all projects onto this range, so beyond it none of the points are relevant in the final analysis.

From the above curve, we can see that when input pattern is (0, 1), the corresponding $f(x) = 1.0$ where $x = 0.73$. Similarly, the other three inputs give the expected correct answers. In the region projected onto between (0.3, 0.9), the slope of the activation is nearly 0 and $f = 1$, and beyond this region, the function slopes down towards 0. Thus, we can see that ADFUNN has learned a fuzzy XOR.

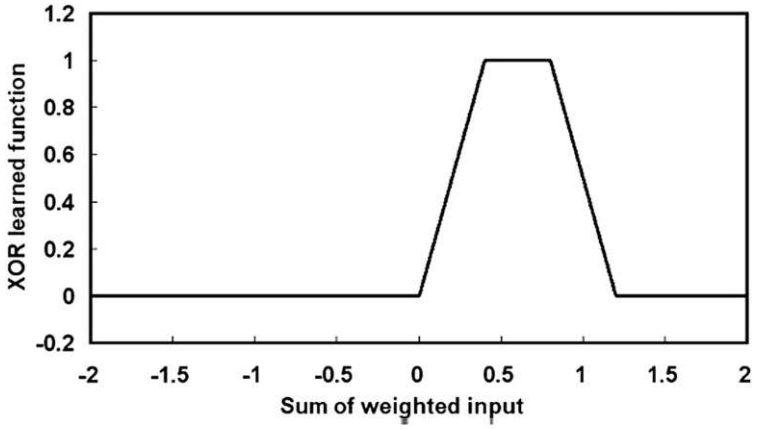

Fig. 2. XOR problem solved using ADFUNN.

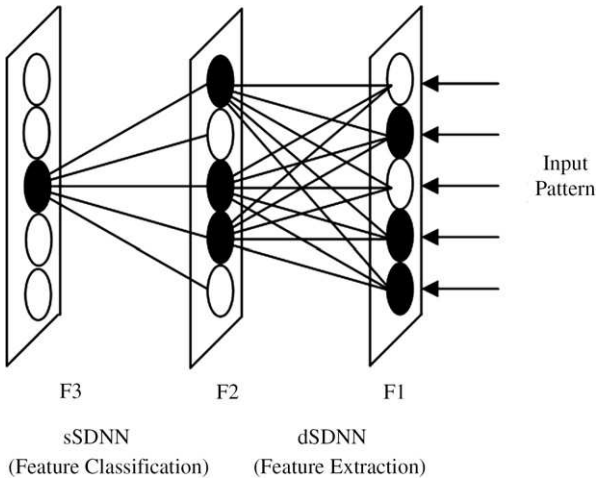

Fig. 3. Snap–drift network.

5. The snap–drift algorithm

This type of network was first introduced by Palmer-Brown and Lee [9–11] is shown in Fig. 3. The first layer, dSDNN learns to group the input patterns according to their features. In this case, 10 F1 nodes (only 3 shown in Fig. 3), whose weight prototypes best match the current input pattern, are used as the input data to the sSDNN module for feature classification. In the dSDNN module, the output nodes with the highest net input are accepted as winners. In the sSDNN module, a quality assurance threshold is introduced. If the net input of a sSDNN node is above the threshold, the output node is accepted as the winner; otherwise a new uncommitted output node will be selected as the new winner and initialised with the current input pattern. In general terms, the snap–drift algorithm can be stated as: $w = \alpha\,(\text{snap}) + \sigma\,(\text{drift})$, where α and σ are toggled between (0, 1) and (1, 0) at the end of each epoch. The point of this is to perform two complementary forms of feature discovery within one system.

In this study the neural network is unsupervised snap–drift (SDNN). One of the strengths of the SDNN is the ability to adapt rapidly in a non-stationary environment where new patterns are introduced over time. The learning process utilises a novel algorithm that performs a combination of fast, convergent, minimalist learning (snap) and more cautious learning (drift) to capture both precise sub-features in the data and more general holistic features. Snap and drift learning phases are combined within a learning system (Fig. 3) that toggles its learning style between the two modes.

On presentation of input data patterns at the input layer, the distributed SDNN (dSDNN) will learn to group them according to their features using snap–drift [9–11]. The neurons whose weight prototypes result in them receiving the highest activations are adapted. Weights are normalised weights so that in effect only the angle of the weight vector is adapted, meaning that a recognised feature is based on a particular ratio of values, rather than absolute values. The output winning neurons from dSDNN act as input data to the selection SDNN (sSDNN) module for the purpose of feature grouping and this layer is also subject to snap–drift learning.

The learning process is unlike error minimisation and maximum likelihood methods in MLPs and other kinds of networks which perform optimisation for classification or equivalents by for example pushing features in the direction that minimizes error, without any requirement for the feature to be statistically significant within the input data. In contrast, SDNN toggles its learning mode to find a rich set of features in the data and uses them to group the data into categories.

Each weight vector is bounded by snap and drift: snapping gives the angle of the minimum values (on all dimensions) and drifting gives the average angle of the patterns grouped under the neuron. Snapping essentially provides an anchor vector pointing at the 'bottom left hand corner' of the pattern group for which the neuron wins. This represents a feature common to all the patterns in the group and gives a high probability of rapid (in terms of epochs) convergence (both snap and drift are convergent, but snap is faster). Drifting tilts the vector towards the centroid angle of the group and ensures that an average, generalised feature is included in the final vector. The angular range of the pattern-group membership depends on the proximity of neighbouring groups (competition), but can also be controlled by adjusting a threshold on the weighted sum of inputs to the neurons.

The following is a summary of the steps that occurs in SDNN:

Step 1: Initialise parameters: ($a = 1$, $s = 0$).
Step 2: In each epoch (t) for each input pattern.
  Step 2.1: Find the D ($D = 10$) winning nodes at F2 with the largest net input.
  Step 2.2: Weights of dSDNN adapted according to the alternative learning procedure: (a, s) becomes Inverse (a, s) after every successive epoch.

Step 3: Process the winning nodes as input pattern of sSDNN.
Step 3.1: Find the node at F3 with the largest net input.
Step 3.2: Test the threshold condition:

IF (the net input of the node is greater than the threshold).
Weights of the sSDNN output node adapted according to the alternative learning procedure: (a, s) becomes inverse (a, s) after every successive epoch.
ELSE An uncommitted sSDNN output node is selected and its weights are adapted according to the alternative learning procedure: (a, s) becomes Inverse (a, s) after every successive epoch.

6. Snap–drift adaptive function neural network (SADFUNN) for optical and pen-based recognition of handwritten digits

By combining two modal learning methods of an unsupervised snap–drift and a supervised ADFUNN together, SADFUNN is shown in Fig. 4. Input patterns are introduced at the input layer F1, the distributed SDNN (dSDNN) learns to group them. The winning F2 nodes, whose prototypes best match the current input pattern, are used as the input data to ADFUNN. For each output class neuron in F3, there is a linear piecewise function. Functions and weights are adapted in parallel. We obtain the output error and adapt the two nearest *f*-points separately, using a function modifying version of the delta rule on a proximal-proportional basis, as in Section 4.1. Patterns are being applied to ADFUNN after snap–drift has converged. Because ADFUNN can only optimise once snap–drift has successfully extracted the features.

6.1. Optical and pen-based recognition of handwritten digits datasets

These two complex cursive-letter datasets are those of handwritten digits presented by Alpaydin et al. [1,2]. They are two different representations of the same handwritten digits. 250 samples per person are collected from 44 people who filled in forms which were then randomly divided into two sets: 30 forms for training and 14 forms by distinct writers for writer-independent testing.

As seen in Fig. 5, the optical data was generated by using the set of programs available from NIST [6] to extract normalized bitmaps of handwritten digits from a pre-printed form. Its representation is a static image of the pen tip movement that have occurred as in a normal scanned image. It is an 8×8 matrix of elements in the range of 0–16 which gives 64 dimensions. There are 3823 training patterns and 1797 writer-independent testing patterns in this dataset.

The pen-based dataset is a dynamic representation of the movement of the pen as the digit is written on a pressure-sensitive tablet. It is generated by a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The raw data consists of integer values between 0 and 500 at the tablet input box resolution, and they are normalised to the range 0–100. This dataset's representation has eight(x, y) coordinates and thus 16 dimensions are needed. There are 7494 training patterns and 3498 writer-independent testing patterns.

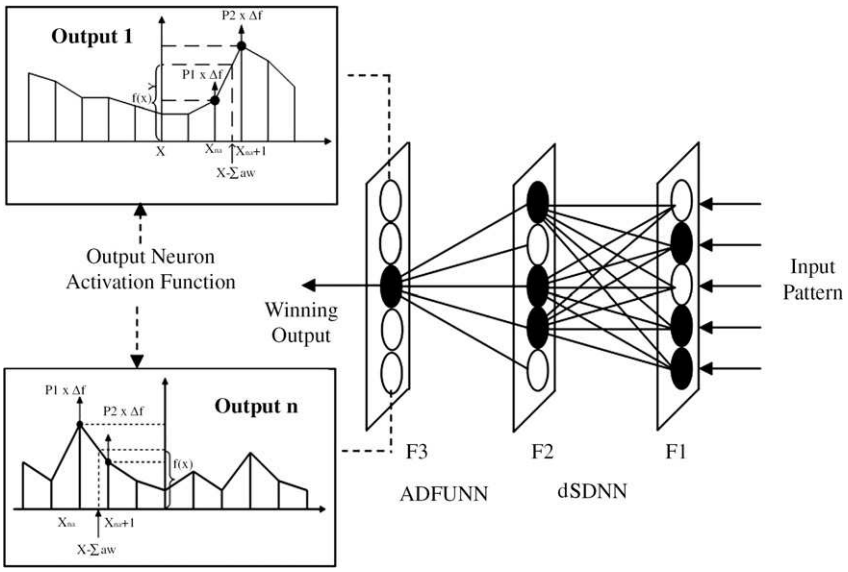

Fig. 4. Architecture of the SADFUNN network.

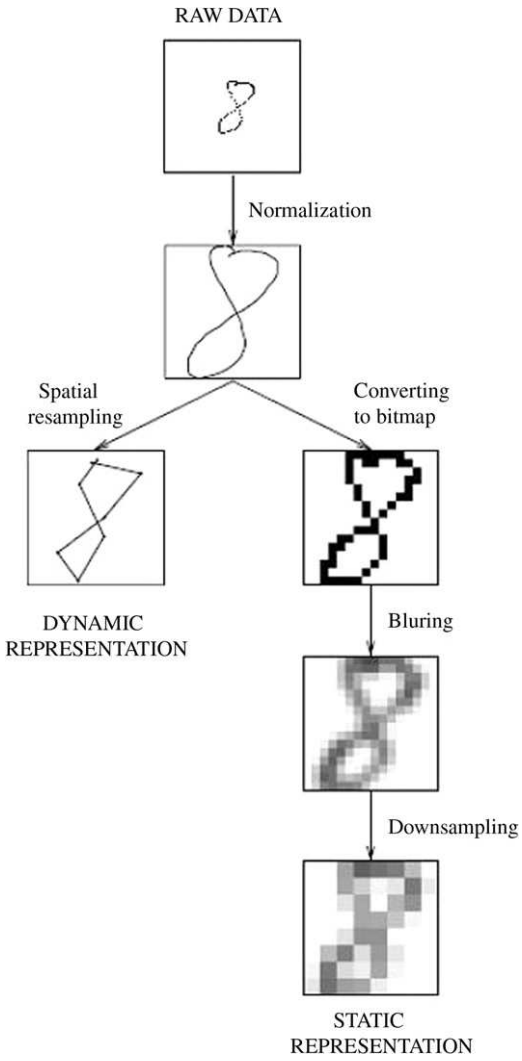

Fig. 5. The processing of converting the dynamic (pen-based) and static (optical) representations.

6.2. Snap–drift adaptive function neural network (SADFUNN) for optical and pen-based recognition of handwritten digits

In ADFUNN, the activation functions are adapted in parallel with the weights using a function modifying version of delta rule. If snap–drift and ADFUNN run at the same time, the initial learning in ADFUNN will be redundant, since it can only optimise once snap–drift is stable, so ADFUNN learns after snap–drift has converged.

All the inputs are scaled from the range of {0, 16} to {0, 1} for the optical dataset and from {0, 100} to {0, 1} for the pen-based dataset. Training patterns are passed to the snap–drift network for feature extraction. After a couple of epochs (feature extraction learned very fast in this case, although 7494 patterns need to be classified), the learned dSDNN is ready to supply ADFUNN for pattern recognition. The training patterns are introduced to dSDNN again but without learning. The winning F2 nodes (whose prototypes best match the current input pattern) are used to form the input data for ADFUNN.

In this single layer ADFUNN, the 10 digits are the output classes. Weights are initialised to 0. *F*-points are initialised to 0.5. Each *F*-point is simply the value of the activation function for a given input sum. *F*-points are equally spaced, and the function value between points is on the straight line joining them. A weight limiter is also applied to ensure that the adaptation to weights will not be too large in order to maintain the stability. The two learning rates FL and WL are equal to 0.1 and 0.000001 respectively. These training patterns' $\sum awj$ has a known range of $[-10, 10]$. It has a precision of 0.01, so 2001 points encode all training patterns for output.

Now the network is ready to learn using the general learning rule of ADFUNN outlined in Section 4.1. By varying the number of snap–drift neurons (learned features) and winning neurons (detected features) in F2, within 200 epochs in each run, about 99.53% and 99.2% correct classification can be achieved for the training data of the optical and pen-based datasets

respectively. We get the output neuron functions in Figs. 6 and 7 (only a few learned functions listed here due to space limitation):

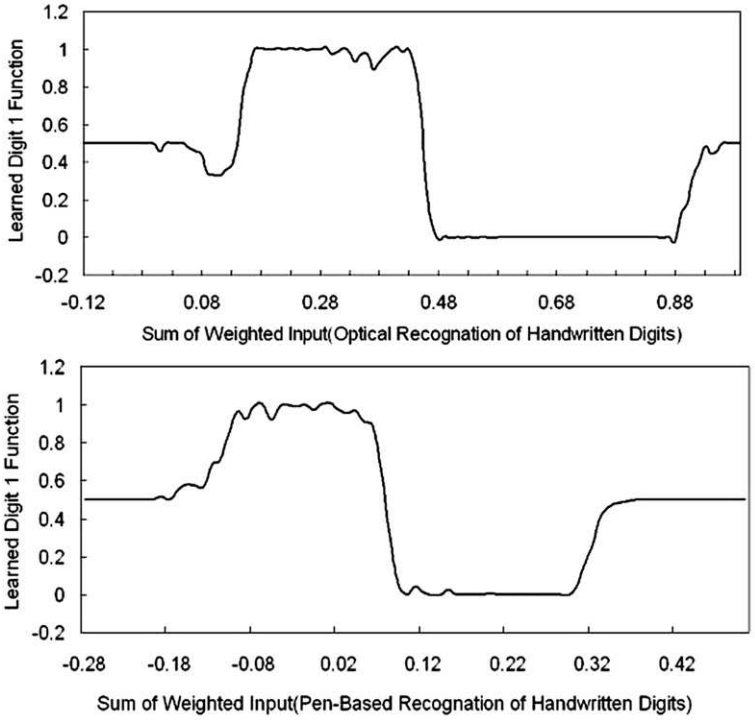

Fig. 6. Digit 1 learned functions in optical dataset and in pen-based dataset using SADFUNN.

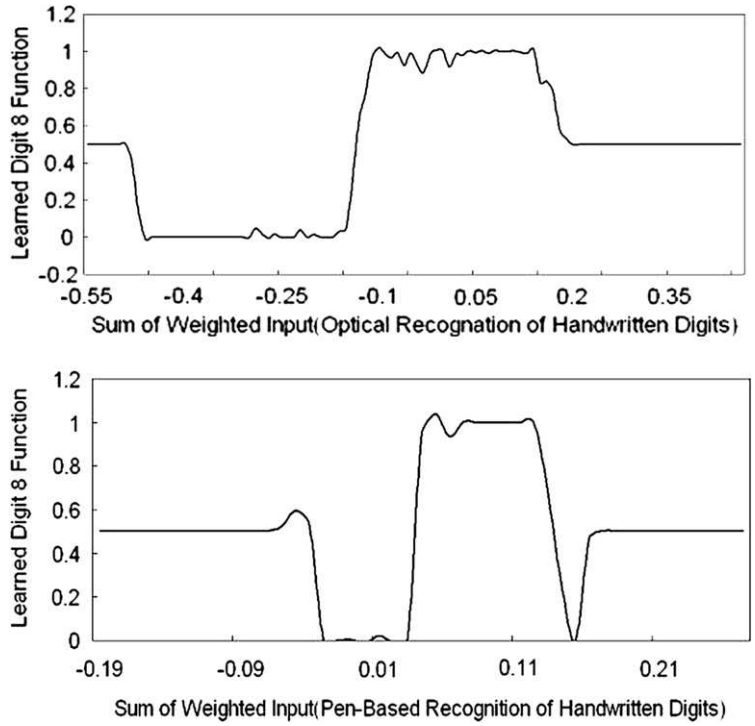

Fig. 7. Digit 8 learned functions in optical dataset using and in pen-based dataset using SADFUNN.

7. Results

We test our network using the two writer-independent testing data for both of the optical recognition and pen-based recognition tasks and the results are in Fig. 8. Performance varies with a small number of parameters, including learning rates FL, WL, the number of snap–drift neurons (features) and the number of winning features.

A large total number of features has a positive effect on the overall performance, however too many may limit generalisation if there is too much memorisation. The above performance charts show how the generalisation changes along with the total number of features.

The following are examples of some misclassified patterns from SADFUNN for optical recognition case.

As we can see in Fig. 9 a digit 9 pattern was misclassified as a 5. The upper part of digit 5 is almost looped, making the 5 similar to a 9.

Fig. 10 illustrates similar situations with confusions based on identifiable similarities.

8. Related work

Using multistage classifiers involving a combination of a rule-learner MLP with an exception-learner *k*-NN, the authors of the two datasets reported 94.25% and 95.26% accuracy on the writer-independent testing data for optical recognition and

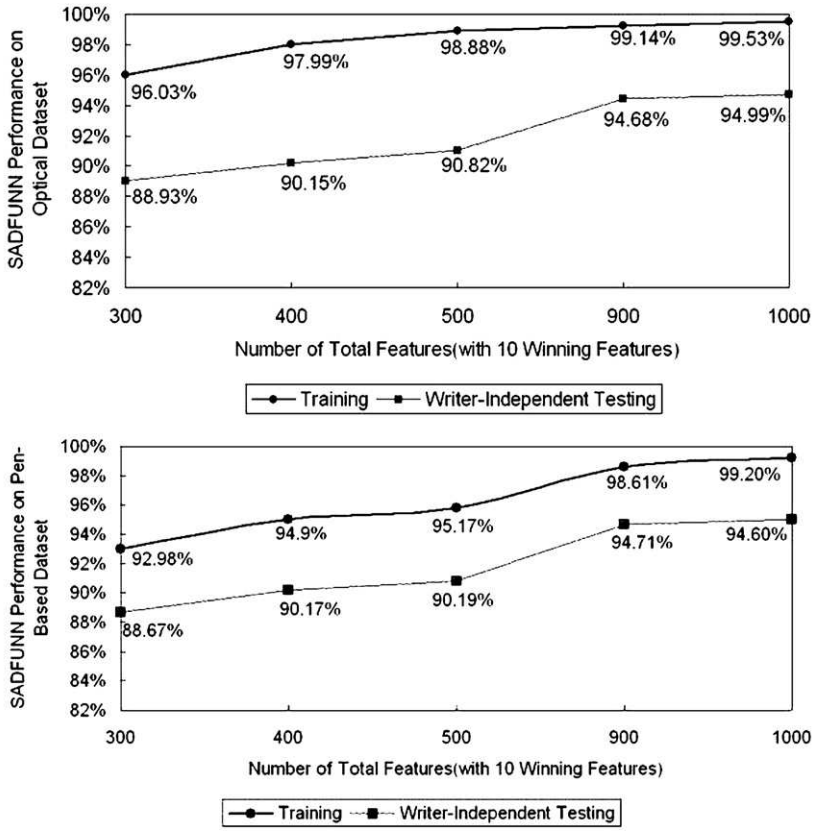

Fig. 8. The performances of training and testing for optical dataset and for pen-based dataset using SADFUNN.

Digit	0	1	2	3	4	5	6	7	8	9
Actual output	0.0	0.0	0.0	0.11	0.0	**0.82**	0.0	0.0	0.0	**0.35**
Expected output	0	0	0	0	0	0	0	0	0	1

Fig. 9. Digit 9 misclassified to digit 5.

Digit	0	1	2	3	4	5	6	7	8	9
Actual output	0.0	0.0	**0.49**	0.0	0.0	0.0	0.0	0.12	**0.52**	0.1
Expected output	0	0	1	0	0	0	0	0	0	0

Digit	0	1	2	3	4	5	6	7	8	9
Actual output	0.0	0.0	0.0	**0.25**	0.0	0.0	0.0	0.1	0.0	**0.82**
Expected output	0	0	0	1	0	0	0	0	0	0

Fig. 10. Digit 2 misclassified to digit 8 and digit 3 misclassified to digit 9.

pen-based recognition datasets respectively [1,2]. Patterns are passed to a MLP with 20 hiddens, and all the rejected patterns are passed to a k-nearest neighbours with $k = 9$ for a second phase of learning.

For the optical recognition task, 23% of the writer-independent test data are not classified by the MLP. They will be passed to k-NN to give a second classification. In our single network combination of a single layer snap–drift and a single layer ADFUNN (SADFUNN) only 5.01% patterns were not classified on the testing data. SADFUNN proves to be a highly effective network with fast feature extraction and pattern recognition ability. Similarly, with the pen-based recognition task, 30% of the writer-independent test data are rejected by the MLP, whereas only 5.4% of these patterns were misclassified by SADFUNN.

Their original intention was to combine multiple representations (dynamic pen-based recognition data and static optical recognition data) of a handwritten digit to increase classification accuracy without increasing the system's complexity and recognition time. By combing the two datasets, they get 98.3% accuracy on the writer-independent testing data. However, we don't experiment with this on SADFUNN because they have already proved the combination of multiple presentations work better than a single one, and also because SADFUNN has already exhibited extremely high generalisation ability compared to a MLP, and it is easy and fast to train and implement.

9. Conclusions

In this paper, we explored unsupervised snap–drift combined with a supervised ADFUNN acting on the activation functions, to perform classification. snap–drift proves very effective in extracting distinct features from the complex cursive-letter datasets. Experiments show only a couple of epochs are enough for the feature discovery. The supervised single layer ADFUNN solves these linearly inseparable feature classification problems rapidly without hidden neurons. From the results, it is very clear that in combination within one network (SADFUNN), these two methods exhibited higher and more computationally efficient generalisation abilities than MLPs.

References

[1] F. Alimoglu, E. Alpaydin, Combining multiple representations for pen-based handwritten digit recognition, Turkish Journal of Electrical Engineering and Computer Sciences 9 (1) (2001) 1–12.
[2] E. Alpaydin, C. Kaynak, F. Alimoglu, Cascading multiple classifiers and representations for optical and pen-based handwritten digit recognition, in: The 7th International Workshop in Frontiers in Handwritting Recognition (IWFHR00), Amsterdam, The Netherlands, September 2000.
[3] E. Alpaydin, F. Alimoglu for Optical Recognition of Handwritten Digits and E. Alpaydin, C. Kaynak for Pen-Based Recognition of Handwritten Digits http://www.ics.uci.edu/~mlearn/databases/optdigits/ <http://www.ics.uci.edu/~mlearn/databases/pendigits/>.
[4] A. Çelikyılmaz, I.B. Türkşen, Fuzzy functions with support vector machines, Information Sciences 177 (23) (2007) 5163–5177.
[5] B. Chakraborty, G. Chakraborty, A new feature extraction technique for on-line recognition of handwritten alphanumeric characters, Information Sciences 148 (1-4) (2002) 55–70.
[6] M.D. Garris et al., NIST form-based handprint recognition system, National Institute of Standard and Technology Interagency Reports (NISTIRs) 5469, 1991.
[7] M. Kang, D. Palmer-Brown, An adaptive function neural network (ADFUNN) for phrase recognition, in: The International Joint Conference on Neural Networks (IJCNN05), Montréal, Canada, 2005.
[8] T. Kohonen, Improved versions of learning vector quantization, in: Proceeding of International Joint Conference on Neural Networks(IJCNN'90), 1990, pp. 545–550.
[9] S.W. Lee, D. Palmer-Brown, C.M. Roadknight, Performance-guided neural network for rapidly self-organising active network management (Invited Paper), Journal of Neurocomputing 61C (2004) 5–20.
[10] S.W. Lee, D. Palmer-Brown, Phonetic feature discovery in speech using snap–drift, in: International Conference on Artificial Neural Networks (ICANN'06), Athens, Greece, 10th–14th September 2006, pp. 952–962.
[11] S.W. Lee, D. Palmer-Brown, Modal learning in a neural network, in: 1st Conference in Advances in Computing and Technology (ACT06), London, United Kingdom, 24th January 2006, pp. 42–47.
[12] D. Palmer-Brown, M. Kang, ADFUNN: An adaptive function neural network, in: The 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA05), Coimbra, Portugal, 2005.

[13] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychological Review 65 (1958) 386–408.
[14] G. Scheler, Regulation of neuromodulator efficacy: implications for whole-neuron and synaptic plasticity, Progress in Neurobiology 72 (6) (2004).
[15] G. Scheler, Memorization in a neural network with adjustable transfer function and conditional gating, Quantitative Biology 1 (2004).
[16] R.R. Yager, An extension of the Naive Bayesian classifier, Information Sciences 176 (5) (2006) 577–588.
[17] J. Zhang, Z. Li, Adaptive nonlinear auto-associative modeling through manifold learning, in: 9th Pacific–Asia Conference (PAKDD05), Hanoi, Vietnamn, 2005, pp. 599–604.