Snap-drift ADaptive FUnction Neural Network (SADFUNN) for Optical and Pen-Based Handwritten Digit Recognition

Miao Kang, Dominic Palmer-Brown
University of East London, UK
M.Kang@uel.ac.uk, D.Palmer-Brown@uel.ac.uk

Abstract

An ADaptive Function Neural Network (ADFUNN) is combined with the on-line snap-drift learning method in this paper to solve an Optical Recognition of Handwritten Digits problem and a Pen-Based Recognition of Handwritten Digits problem. Snap-Drift [1] employs the complementary concepts of minimalist learning (snap) and drift (towards the input patterns) learning, and is a fast unsupervised method suitable for on-line learning and/or non-stationary environments where new patterns are continually introduced. The ADaptive FUction Neural Network (ADFUNN) presented in this paper [2, 3] is based on a linear piecewise neuron activation function that is modified by a novel gradient descent supervised learning algorithm. It has previously been applied to the Iris dataset, and a natural language phrase recognition problem, exhibiting impressive generalisation classification ability with no hidden neurons [2, 3]. The unsupervised single layer Snap-Drift is effective in extracting distinct features from these 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 with the Optical Recognition of Handwritten Digits and the Pen-Based Recognition of Handwritten Digits problems [4] from UCI repository. The problems are learned rapidly and higher generalisation results are achieved than a MLP.

1. Motivation

Artificial neural network learning is typically accomplished via adaptation between neurons. This paper describes adaptation that is 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 this approach 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 [5, 6] 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 firstly introduced by Palmer-Brown and Lee[1, 7 and 8] 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[7, 8]. Since then, the snap-drift algorithm has proved invaluable for continuous learning in many applications. In a Snap-Drift network, snap is based on the logical intersection method from ART, which is implemented here as a fuzzy AND; and drift is based on Learning Vector Quantization (LVQ) [9]. Snap-drift harnesses the complementary strengths of the two forms of learning which are dynamically combined in a rapid form of adaptation that balances minimalist pattern intersection learning with Learning Vector Quantization. This 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. A single layer adaptive function network (ADFUNN)

We provide 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 figure 1.

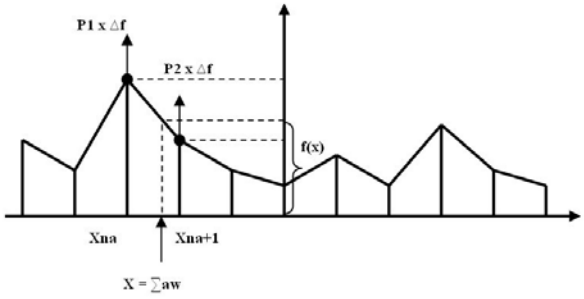

Figure 1. Adapting the linear piecewise neuronal activation function in ADFUNN

We calculate ∑aw, and find the two neighbouring f-points that bound ∑aw. Two proximal f-points are adapted separately, on a proximal-proportional basis. The proximal-proportional value P1 is (Xna+1 - x)/(Xna+1 – Xna) and value P2 is (x - Xna)/(Xna+1 - Xna). 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 2.1 to calculate Δf.

2.1. 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.

Step1: calculate output error, E, for input, A.
Step2: adapt weights to each output neuron:
$\Delta w = WL \times Fslope \times A \times E$
$w' = w + \Delta w$
weights normalisation
Step3: adapt function for each output neuron:
$\Delta f (\sum aw) = FL \times E$
$f'_1 = f_1 + \Delta f \times P1$, $f'_2 = f_2 + \Delta f \times P2$
Step4: $f (\sum aw) = f' (\sum aw)$;
$w = w'$.
Step5: randomly select a pattern to train
Step6: repeat step 1 to step 5 until the output error tends to a steady state.

2.2. 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 initialised 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 figure 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.

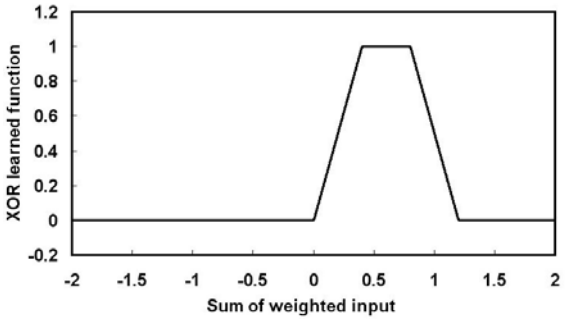

Figure 2. XOR problem solved using ADFUNN

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.

3. The Snap-Drift Algorithm

This type of network was first introduced by Palmer-Brown and Lee [1, 7, and 8] 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 whose weight prototypes best match the current input pattern, are used as the input data to the sSDNN module for feature classification (only 3 winners shown in figure 3). 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: $w = \alpha(snap) + \sigma(drift)$, 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 (new candidate road attributes in this case) 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 (Figure 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 [1, 7 and 8]. 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.

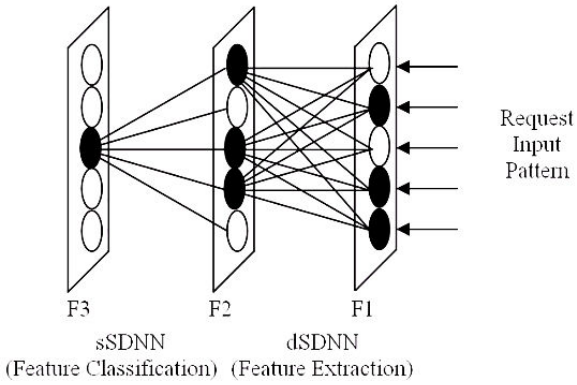

Figure 3. Snap-Drift Neural Network (SDNN) architecture (Palmer-Brown & Lee [1, 7 and 8])

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.

4. Snap-drift ADaptive FUnction Neural Network (SADFUNN) on Optical and Pen-Based Recognition of Handwritten Digits

SADFUNN is shown in figure 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 and 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.

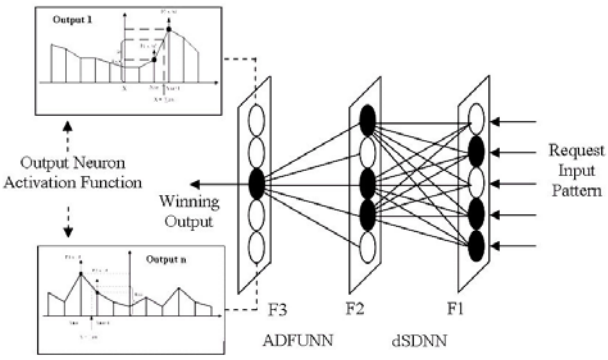

Figure 4. Architecture of the SADFUNN network

4.1. 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 [10, 11]. 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 test.

The optical one was generated by using the set of programs available from NIST [12] 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 x 8 matrix of elements in the range of 0 to 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 where 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 to 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.

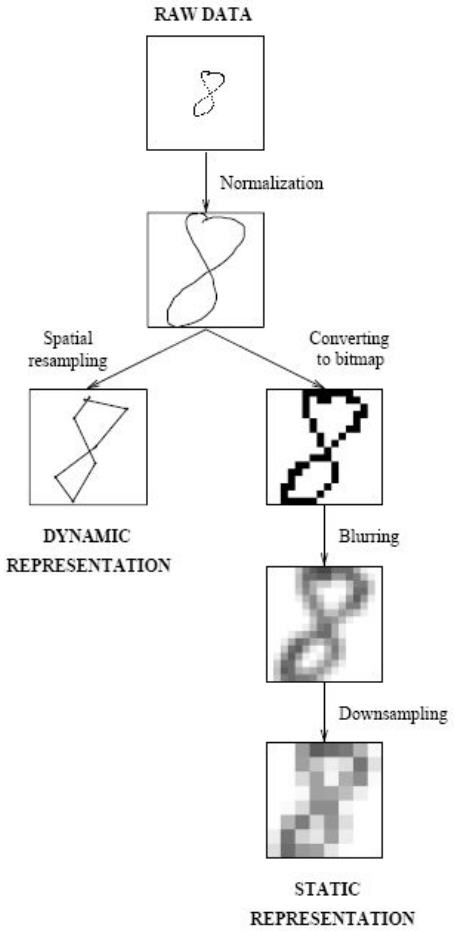

Figure 5. The processing of converting the dynamic (pen-based) and static (optical) representations

4.2. Snap-drift ADaptive FUnction Neural Network (SADFUNN) on Optical and Pen-Based Recognition of Handwritten Digits

In ADFUNN, weights and activation functions are adapted in parallel 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 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 for best learning results. 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, but every 250 samples are from the same writer, many similar samples exist), 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 as the input data to 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 ensure 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 2.1. By varying the number of snap-drift neurons (features) and winning features number in F2, within 200 epochs in each run, about 99.53% and 99.2% correct classifications for the best can be achieved for the training data for the optical and pen-based datasets respectively. We get the following output neuron functions (only a few learned functions listed here due to space limitation):

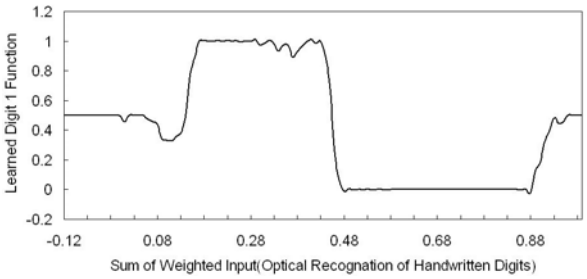

Figure 6. Digit 1 learned function in optical dataset using SADFUNN

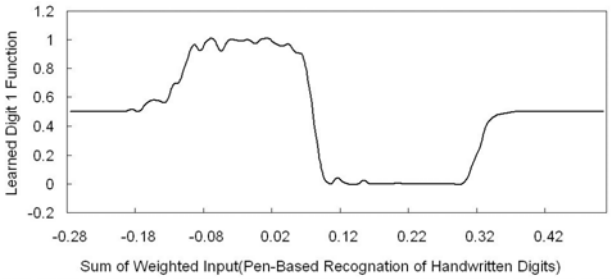

Fig. 7. Digit 1 learned function in pen-based dataset using SADFUNN

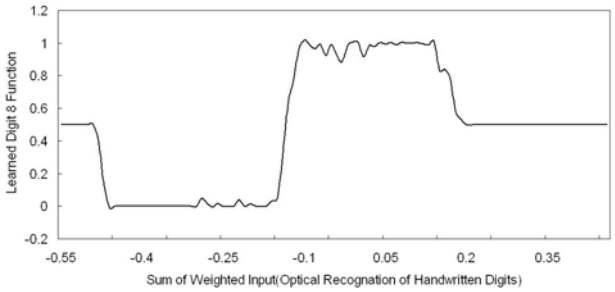

Fig. 8. Digit 8 learned function in optical dataset using SADFUNN

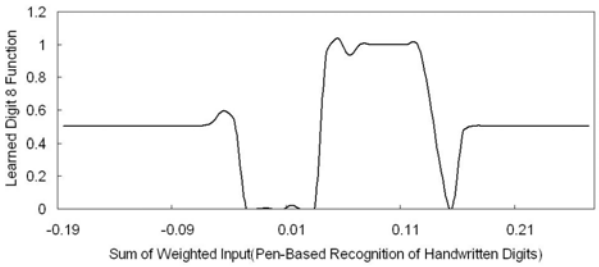

Fig. 9. Digit 8 learned function in pen-based dataset using SADFUNN

5. Results

We test our network using the two writer-independent testing data for both of the optical recognition and pen-based recognition tasks. 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.

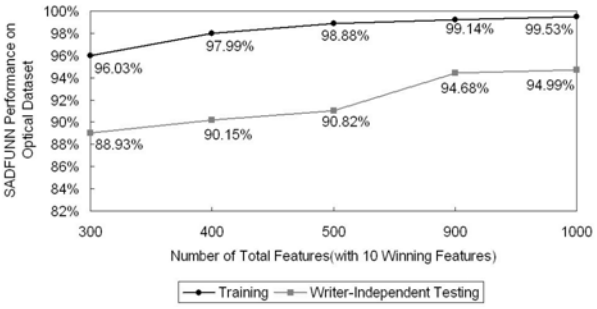

Fig. 10. The performance of training and testing for optical dataset using SADFUNN

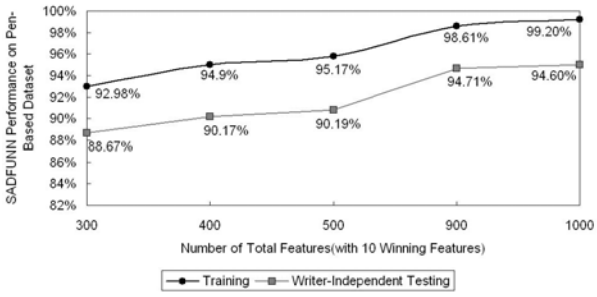

Fig. 11. The performance of training and testing for pen-based dataset using SADFUNN

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.

6. 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 pen-based recognition datasets respectively [10, 11]. 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) network 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 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.

Zhang and Li [13] propose an adaptive nonlinear auto-associative modelling (ANAM) based on Locally Linear Embedding (LLE) for learning both intrinsic principal features of each concept separately. 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.

7. Conclusions

In this paper, we explored unsupervised Snap-Drift combined with a supervised ADFUNN acting on the activation functions alone, to perform classification. Snap-Drift is very effective in extracting distinct features from the complex cursive-letter datasets. Experiments show only a couple of epochs are enough for the feature classification. It helps the supervised single layer ADFUNN to solve these linearly inseparable problems rapidly without any hidden

neuron. From the experimental results, it is clear that when combined within one network (SADFUNN), the two methods exhibited higher generalisation abilities than MLPs even though the learning process is simpler.

8. References

[1] S. W. Lee, D. Palmer-Brown and C. M. Roadknight, "Performance-guided Neural Network for Rapidly Self-Organising Active Network Management (Invited Paper)", *Journal of Neurocomputing*, 61C, 2004, pp. 5 – 20.

[2] D. Palmer-Brown and M. Kang, "ADFUNN: An adaptive function neural network", *the 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA05)*, Coimbra, Portugal, 2005, pp. 1-4.

[3] M. Kang and D. Palmer-Brown, "An Adaptive Function Neural Network (ADFUNN) for Phrase Recognition", *the International Joint Conference on Neural Networks (IJCNN05)*, Montréal, Canada, 2005, pp593-597.

[4] 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/

[5] G. Scheler, "Regulation of neuromodulator efficacy: Implications for whole-neuron and synaptic plasticity", *Progress in Neurobiology*, Vol.72, No.6, 2004.

[6] G. Scheler, "Memorization in a neural network with adjustable transfer function and conditional gating", *Quantitative Biology*, Vol.1, 2004.

[7] S. W. Lee and D. Palmer-Brown, "Phonetic Feature Discovery in Speech using Snap-Drift. International Conference on Artificial Neural Networks", *ICANN'2006, Athens, Greece,* 10th - 14th September 2006, pp. 952 – 962.

[8] S. W. Lee and D. Palmer-Brown, "Modal Learning in A Neural Network", *1st Conference in Advances in Computing and Technology*, London, United Kingdom, 24th January 2006, pp. 42 - 47.

[9] T. Kohonen, "Improved Versions of Learning Vector Quantization", *Proc. IJCNN'90,* 1990, pp.545 – 550.

[10] E. Alpaydin, C. Kaynak, F. Alimoglu, "Cascading Multiple Classifiers and Representations for Optical and Pen-Based Handwritten Digit Recognition", *IWFHR,* Amsterdam, The Netherlands, September 2000.

[11] F. Alimoglu, E. Alpaydin, "Combining Multiple Representations for Pen-based Handwritten Digit Recognition", *ELEKTRIK: Turkish Journal of Electrical Engineering and Computer Sciences*, **9**(1), 2001, pp.1-12.

[12] M.D. Garris et al, NIST Form-Based Handprint Recognition System, *NISTIR* 5469, 1991.

[13] J. Zhang and Z. Li, "Adaptive Nonlinear Auto-Associative Modeling through Manifold Learning", *PAKDD,* 2005, pp.599-604.