Chapter 9

Large Scale Memory Storage and Retrieval (LAMSTAR) Network

9.1. Motivation

The neural network discussed in the present section is an artificial neural network for large scale memory storage and retrieval of information [Graupe and Kordylewski, 1996a,b]. This network attempts to imitate, in a gross manner, processes of the human central nervous system (CNS), concerning storage and retrieval of patterns, impressions and sensed observations, including processes of forgetting and of recall. It attempts to achieve this without contradicting findings from physiological and psychological observations, at least in an input/output manner. Furthermore, the LAMSTAR (LArge Memory STorage And Retrieval) model considered attempts to do so in a computationally efficient manner, using tools of neural networks from the previous sections, especially *SOM* (Self Organizing Map)-based network modules (similar to those of Sec. 8 above), combined with statistical decision tools. The LAMSTAR network is therefore *not a specific network but a system of networks* for storage, recognition, comparison and decision that facilitates such storage and retrieval to be accomplished. It combines Kohonen's WTA (Winner-Take-All) principle as in Chap. 8 (Kohonen, 1984), with Link Weights in the sense of Hebb's principle (Hebb, 1949) of Sec. 3.1 and of the Pavlovian dog (Pavlov, 1927) example of the same Section. These Link Weights thus allow efficient integration of (very) many SOM layers in the LAMSTAR network.

The Link Weights are conceptually based on the Kantian principle of "Verbindungen", namely, "Interconnections" introduced in his famous "Critique of Pure Reason (Kant, 1781 — also see: Ewing, 1938). According to Kant, understanding is based on two concepts, memory elements ("things") and interconnections between them. Without both, Understanding is not possible. In artificial neural networks (ANN's). memory storage is facilitated via, say Associative Memory weights, as in the Hopfiled NN (Chap. 7 above) or the Kohonen SOM layers (Chap. 8 above), while Verbindungen are facilitated via Hebb's principle (Hebb, 1949), which is only implicit in most designs, as in the previous in the chapters. In the LAMSTAR NN, the "Verbindungen" are introduced in a purely Hebbian and even Pavlovian manner (Pavlov, 1927), through the use of Link-Weights (Graupe and Kordylewski, 1996b).

These link weights are those observed by functional MRI as connections (flow) of neural information from one section of the Central Nervous System (CNS) to another. They are related to the address-correlation gates (links) in (Graupe and Lynn, 1969) and to Minsky's K-Lines (Knowledge-Lines), as in (Minsky, 1980). The use of Link-Weights makes the LAMSTAR ANN a transparent network, in contrast to other ANN's, noting that the lack of transparency was one of the very main criticisms of ANN.

9.2. Basic Principles of the LAMSTAR Neural Network

The LAMSTAR neural network is specifically designed for application to retrieval, diagnosis, classification, prediction and decision problems which involve a very large number of categories. The resulting LAMSTAR neural network [Graupe, 1997, Graupe and Kordylewski, 1998] is designed to store and retrieve patterns in a computationally efficient manner, using tools of neural networks, especially Kohonen's SOM (Self Organizing Map)-based network modules [Kohonen, 1988], combined with statistical decision tools.

By its structure as described in Sec. 9.2, the LAMSTAR network is uniquely suited to deal with analytical and non-analytical problems where data are of many vastly different categories and vector-dimensions, where some categories may be missing, where data are both exact and fuzzy and where the vastness of data requires very fast algorithms [Graupe, 1997, Graupe and Kordylewski, 1998]. These features are rare to find, especially when coming together, in other neural networks.

The LAMSTAR can be viewed as in intelligent expert system, where expert information is continuously being ranked for each case through learning and correlation. What is unique about the LAMSTAR network is its capability to deal with non-analytical data, which may be exact or fuzzy and where some categories may be missing. These characteristics are facilitated by the network's features of forgetting, interpolation and extrapolation. These allow the network to zoom out of stored information via forgetting and still being able to approximate forgotten information by extrapolation or interpolation. The LAMSTAR was specifically developed for application to problems involving very large memory that relates to many different categories (attributes), where some of the data is exact while other data are fuzzy and where (for a given problem) some data categories may occasionally be totally missing. Also, the LAMSTAR NN is insensitive to initialization and is doe not converge to local minima. Furthermore, in contrast to most Neural Networks (say, Back-Propagation as in Chap. 6), the LAMSTAR's unique weight structure makes it fully transparent, since its weights provide clear information on what is going on inside the network. Consequently, the network has been successfully applied to many decision, diagnosis and recognition problems in various fields.

The major principles of neural networks (NN's) are common to practically all NN approaches. Its elementary neural unit or cell (neuron) is the one employed in all NN's, as described in Chaps. 2 and 4 of this text. Accordingly, if the p inputs into a given neuron (from other neurons or from sensors or transducers at the input to the whole or part of the whole network) at the j'th SOM layer are denoted as $x(ij); i = 1, 2, \ldots, p$, and if the (single) output of that neuron is denoted as y, then the neuron's output y satisfies;

$$y = f\left[\sum_{i=1}^{p} w_{ij} x_{ij}\right] \tag{9.1}$$

where $f[.]$ is a nonlinear function denoted as Activation Function, that can be considered as a (hard or soft) binary (or bipolar) switch, as in Chap. 4 above. The weights w_{ij} of Eq. (9.1) are the weights assigned to the neuron's inputs and whose setting is the learning action of the NN. Also, neural firing (producing of an output) is of all-or-nothing nature [McCulloch and Pitts, 1943]. For details of the setting of the storage weights (w_{ij}), see Secs. 9.3.2 and 9.3.6 below.

The WTA (Winner-Take-All) principle, as in Chap. 8, is employed [Kohonen, 1988], such that an output (firing) is produced ***only*** at the winning neuron, namely, at the output of the neuron whose storage weights w_{ij} are closest to vector $\underline{\mathbf{x}}(j)$ when a best-matching memory is sought at the j'th SOM module.

By using a link weights structure for its decision and browsing, the LAMSTAR network considers not just the stored memory values $w(ij)$ as in other neural networks, but also the interrelations (the Kantian Verbindungen discussed above) between these memories and the decision modules and between the memories themselves. These relations (link weights) are fundamental to its operation. As mentioned above, by Hebb's Law [Hebb, 1949], interconnecting inter-synaptic weights (link weights) adjust and serve to establish flow of neuronal-signal traffic between groups of neurons, such that when a certain neuron fires very often in close time (regarding a given situation/task), then the interconnecting link-weights (not the memory-storage weights) increase as compared to other interconnections. Indeed, link weights serve as Hebbian inter-synaptic weights and adjust accordingly. These weights and their method of adjustment (based on flow of traffic in the interconnections), fit results on CNS organization [Levitan *et al.*, 1997]. They are also responsible to the LAMSTAR's ability to interpolate/extrapolate and operate (with no re-programming or retraining) with incomplete data sets.

9.3. Detailed Outline of the LAMSTAR Network

9.3.1. *Basic structural elements*

The basic storage modules of the LAMSTAR network are modified Kohonen SOM modules [Kohonen, 1988] of Chap. 8 that are Associate-Memory-based WTA, in accordance to degree of proximity of **storage weights** in the BAM-sense to any **input subword** that is being considered per any given **input word** to the NN. In the LAMSTAR network the information is stored and processed via correlation links between individual neurons in separate SOM modules. Its ability to deal with a large number of categories is partly due to its use of simple calculation of **link weights** and by its use of **forgetting** features and features of recovery from forgetting. The link weights are the main engine of the network, connecting many layers of SOM modules such that the emphasis is on (co)relation of link weights between atoms of memory, not on the memory atoms (BAM weights of the SOM modules) themselves. In this manner, the design becomes closer to knowledge processing in the biological central nervous system than is the practice in most conventional artificial neural networks. The forgetting feature too, is a basic feature of biological networks whose efficiency depends on it, as is the ability to deal with incomplete data sets.

The **input word** is a coded real matrix $\boldsymbol{X}$ given by:

$$\underline{X} = \left[\underline{x}_1^T, \underline{x}_2^T, \ldots, \underline{x}_N^T\right]^T \tag{9.2}$$

where T denotes transposition., $\boldsymbol{x}_i^T$ being subvectors (subwords describing categories or attributes of the input word). Each subword $\underline{\boldsymbol{x}}_i$ is channeled to a corresponding i'th SOM module that stores data concerning the i'th category of the input word.

Many input subwords (and similarly, many inputs to practically any other neural network approach) can be derived only after ***pre-processing***. This is the case in signal/image-processing problems, where only autoregressive or discrete spectral/wavelet parameters can serve as a subword rather than the signal itself.

Whereas in most SOM networks [Kohonen, 1988] all neurons of an SOM module are checked for proximity to a given input vector, in the LAMSTAR network only a finite group of p neurons may checked at a time due to the huge number of neurons involved (the large memory involved). The final set of p neurons is determined by link-weights (N_i) as shown in Fig. 9.1. However, if a given problem requires (by considerations of its quantization) only a small number of neurons in a given SOM storage module (namely, of possible states of an input subword), then all neurons in a given SOM module will be checked for possible storage and for subsequent

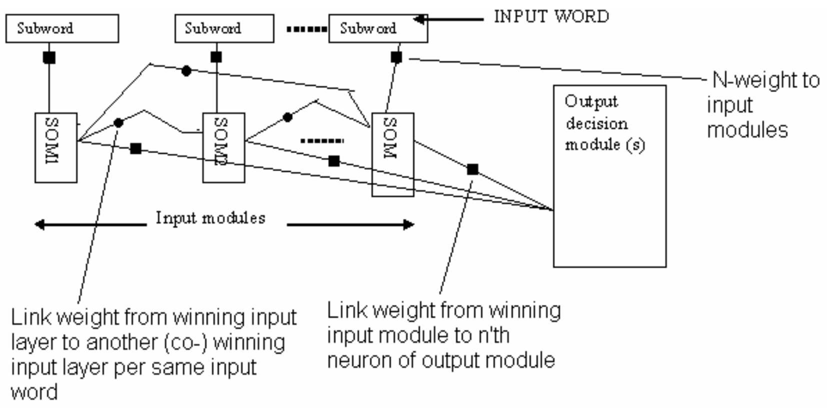

Subword
Subword
Subword
INPUT WORD
N-weight to input modules
SOM1
SOM2
SOM
Output decision module (s)
Input modules
Link weight from winning input layer to another (co-) winning input layer per same input word
Link weight from winning input module to n'th neuron of output module

Fig. 9.1. A generalized LAMSTAR block-diagram.

selection of a winning neuron in that SOM module (layer) and N_i weights are *not* used. Consequently, if the number of quantization levels in an input subword is small, then the subword is channeled directly to all neurons in a predetermined SOM module (layer).

The main element of the LAMSTAR, which forms its decision engine, is the array of **link weights** that interconnect neurons between all input-storage neurons of the input SOM layers and the neurons at the output (decision) layers. These input-layer link weights are updated in accordance with traffic volume. The link weights to the output layers are updated by a reward/punishment process in accordance to success or failure of any decision, thus forming a learning process that is not limited to training data but continuous throughout running the LAMSTAR on a given problem. Weight-initialization is simple and unproblematic as all weight are initially set to zero. The LAMSTAR's feed-forward structure guarantees its stability, since feedback is provided at the end of each cycle, namely at one-step delay. Details on the link weight adjustments, its reinforcement (punishment/reward) feedback policy and related topics are discussed in the sections below.

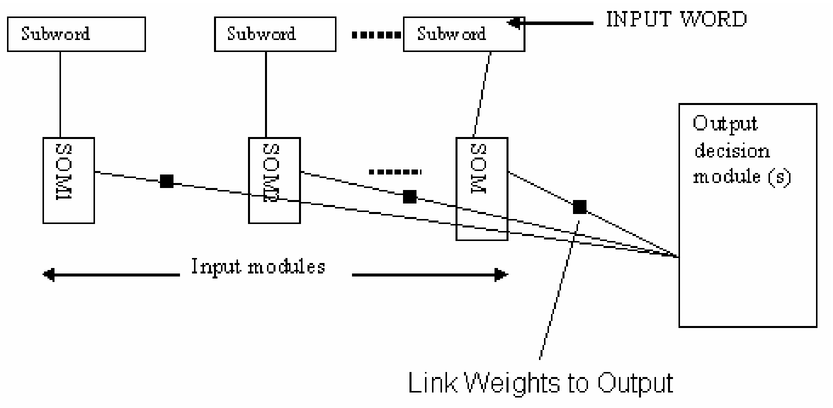


Fig. 9.2. The basic LAMSTAR architecture: simplified version for most applications.

Figure 9.1 gives a block-diagram of the complete and generalized of the LAMSTAR network. A more basic diagram, to be employed in most applications where the number of neurons per SOM layer is not huge, is given in Fig. 9.2. This design is a slight simplification of the generalized architecture. It is also employed in the case studies of Appendices 9.A and 9.B below. Only large browsing/retrieval cases should employ the complete design of Fig. 9.1. In the design of Fig. 9.2, the internal weights from one input layer to other input layers are omitted, as are the N_{ij} weights. Since they are usually not implemented (except for very specific retrieval and search-engine problems from huge databases. Hence, Fig. 9.2 represents the preferred LAMSTAR architecture.

9.3.2. *Setting of input-storage weights and determination of winning neurons*

When a new input word is presented to the system during the training phase, the LAMSTAR network inspects all storage-weight vectors ($\boldsymbol{w}_i$) in SOM module i that corresponds to an input subword $\underline{\mathbf{x}}_i$ that is to be stored. If any stored pattern matches the input subword $\underline{\mathbf{x}}_i$ within a preset tolerance, it is declared as the **winning** neuron for that particularly observed input subword. A **winning** neuron is thus determined for each input based on the similarity between the input (vector $\boldsymbol{x}$ in Figs. 9.1 and 9.2) and a storage-weight vector $\underline{w}$ (stored information). For an input subword $\mathbf{x}_i$, the winning neuron is thus determined by minimizing a distance norm $\| * \|$, as follows:

$$d(j,j) = \|\underline{x}_j - \underline{w}_j\| \leq \|\underline{x}_j - \underline{w}_{k \neq j}\| \triangleq d(j,k) \qquad \forall\, k \tag{9.3}$$

In many application, where storage of purely numerical input subwords is concerned, the storage of such subwords into SOM modules **can be simplified by directly channeling each such subword into a pre-set range of values**, via pre-assigning inequalities for each input-SOM layer. In that case, each range of values will correspond to a given input layer at that SOM. Hence, an input subword whose value is 0.41 will be stored in an input neuron corresponding to a range 0.25 to 0.50, etc... at the given SOM layer, rather than using the algorithm of Eq. (9.3) above.

9.3.3. *Adjustment of resolution in SOM modules*

Equation (9.3), which serves to determine the winning neuron, does not deal effectively with the resolution of close clusters/patterns. This may lead to degraded accuracy in the decision making process when decision depends on local and closely related patterns/clusters which lead to different diagnosis/decision. The local sensitivity of neuron in SOM modules can be adjusted by incorporating an adjustable maximal Hamming distance function $d_{\max}$ as in Eq. (9.4):

$$d_{\max} = \max[d(\underline{x}_i\, \underline{w}_i)]\,. \tag{9.4}$$

Consequently, if the number of subwords stored in a given neuron (of the appropriate module) exceeds a threshold value, then storage is divided into two adjacent storage neurons (i.e. a new-neighbor neuron is set) and $d_{\max}$ is reduced accordingly.

For fast adjustment of **resolution**, link weight to the output layer (as discussed in Sec. 9.3.4 below) can serve to adjust the resolution, such that storage in cells that yield a relatively high N_{ij} weights can be divided (say into 2 cells), while cells with low output link weights can be merged into the neighboring cells. This adjustment can be automatically or periodically changed when certain link weights increase or decrease relative to others over time (and considering the networks forgetting capability as in Sec. 9.3 below).

9.3.4. *Links between SOM modules and from SOM modules to output modules*

Information in the LAMSTAR system is mapped via link weights $L_{i,j}$ (Figs. 9.1, 9.2) between individual neurons in different SOM modules. The LAMSTAR system does not create neurons for an entire input word. Instead, only selected subwords are stored in Associative-Memory-like manner in SOM modules (w weights), and correlations between subwords are stored in terms of creating/adjusting L-links ($L_{i,j}$ in Fig. 9.1) that connect neurons in different SOM modules. This allows the LAMSTAR network to be trained with partially incomplete data sets. The L-links are fundamental to allow interpolation and extrapolation of patterns (when a neuron in an SOM model does not correspond to an input subword but is highly linked to other modules serves as an interpolated estimate). We comment that the setting (updating) of Link Weights, as considered in this sub-section, applies to both link weights between **input-storage** (internal) SOM modules **AND also** link-weights from any storage SOM module and an **output module** (layer). **In most applications it is advisable and economical to consider only links to ouput (decision) modules**. All applications, as in the case studies appended to this chapter, do so.

Specifically, **link weight values** L are **set** (updated) such that for a given input word, after determining a **winning** k'th neuron in module i and a winning m'th neuron in module j, then the link weight $L_{i,j}^{k,m}$ is counted up by a reward increment ΔL, whereas, all other links $L_{i,j}^{s,v}$ may be reduced by a punishment increment ΔM. (Fig. 9.2) [Graupe 1997, Graupe and Kordylewski Graupe, 1997]. The values of L-link weights are modified according to:

$$L_{i,j}^{k,m}(t+1) = L_{i,j}^{k,m}(t) \ + \Delta L : \ L_{i,j}^{k,m} \leq L_{\max} \tag{9.5a}$$

$$L_{i,j}(t+1) = L_{i,j}(t) - \Delta M \tag{9.5b}$$

$$L(0) = 0 \tag{9.5c}$$

where:

$L_{i,j}^{k,m}$: links between winning neuron i in k'th module and winning neuron j in m'th module (which may also be the m'th output module).

ΔL, ΔM: reward/punishment increment values (predetermined fixed values). It is sometimes desirable to set ΔM (either for all LAMSTAR decisions or only when the decision is correct) as:

$$\Delta M = 0 \tag{9.6}$$

$L_{\max}$: maximal links value (not generally necessary, especially when update via forgetting is performed).

The link weights thus serve as address correlations [Graupe and Lynn, 1970] to evaluate traffic rates between neurons [Graupe, 1997, Minsky, 1980]. See Fig. 9.1. The L link weights above thus serve to guide the storage process and to speed it up in problems involving very many subwords (patterns) and huge memory in each such pattern. They also serves to exclude patterns that totally overlap, such that one (or more) of them are redundant and need be omitted. In many applications, the only link weights considered (and updated) are those between the SOM storage layers (modules) and the output layers (as in Fig. 9.2), while link-weights between the various SOM input-storage layers (namely, **internal link-weights**) are **not considered or updated**, unless they are required for decisions related to Sec. 9.3.6 below.

9.3.5. *Determination of winning decision via link weights*

The diagnosis/decision at the output SOM modules is found by analyzing correlation links L between diagnosis/decision neurons in the output SOM modules and the **winning neurons** in all input SOM modules selected and accepted by the process outlined in Sec. 9.3.4. Furthermore, all **L-weight values are set** (updated) as discussed in Sec. 9.3.4 above (Eqs. (9.5) and (9.6)).

The winning neuron (diagnosis/decision) from the output SOM module is a neuron with the highest cumulative value of links L connecting to the selected (winning) input neurons in the input modules. The diagnosis/detection formula for output SOM module (i) is given by:

$$\sum_{k(w)}^{M} L_{k(w)}^{i,n} \geq \sum_{k(w)}^{M} L_{k(w)}^{i,j} \qquad \forall\ i, j, k, n, \quad j \neq n \tag{9.7}$$

where:

i: i'th output module.

n: winning neuron in the i'th output module

$k(w)$: winning neuron in the k'th input module.

module M: number of input modules.

$L_{k(w)}^{i,j}$: link weight between winning neuron in input module k and neuron j in i'th output module.

Link weights may be either positive or negative. They are preferably **initiated** at a small random value close to zero, though initialization of all weights at zero (or at some other fixed value) poses no difficulty. If two or more weights are **equal** then a certain decision must be pre-programmed to be given a priority.

Note that in every input SOM layer there is ONLY one winning neuron (if at all — see Sec. 9.5).

9.3.6. N_j *weights* (*not implemented in most applications*)

The N_j weights of Fig. 9.1 [Graupe and Kordyleski, 1998] are updated by the amount of traffic to a given neuron at a given input SOM module, namely by the accumulative number of subwords stored at a given neuron (subject to adjustments due to forgetting as in Sec. 9.4 below), as determined by Eq. (9.8):

$$\|\underline{x}_i - \underline{w}_{i,m}\| = \min \|\underline{x}_i - \underline{w}_{i,k}\|, \qquad \forall\ k \in \langle l, l+p\rangle; \quad l \sim \{N_{i,j}\} \tag{9.8}$$

where

m: is the winning unit in i'th SOM module (WTA),

$(N_{i,j})$: denoting of the weights to determine the neighborhood of top priority neurons in SOM module i, for the purpose of storage search. In most applications, k covers all neurons in a module and both N_{ij} and l are disregarded, as in Fig. 9.2.

l: denoting the first neuron to be scanned (determined by weights $N_{i,j}$);

$\sim$ denoting proportionality.

The N_j weights of Fig. 9.1 above are only used in huge retrieval/browsing problems. They are initialized at some small random non-zero value (selected from a uniforms distribution) and increase linearly each time the appropriate neuron is chosen as winner.

9.3.7. *Initialization and local minima*

In contrast to most other networks, the LAMSTAR neural network is not sensitive to initialization and will not converge to local minima. All link weights should be initialized with the same constant value, preferably **zero**. However initialization of the storage weights ω_{ij} of Sec. 9.3.2 and of N_j of Sec. 9.3.6, when applicable, should be at random (very) low values.

Again, in contrast to most other neural networks, the LAMSTAR will not converge to a local minimum, due to its link-weight punishment/reward structure since punishments will continue at local minima.

9.4. Forgetting Feature

Forgetting is introduced in by a forgetting factor $F(k)$; such that:

$$L(k+1) = L(k) - F\{k\} \quad \forall\ k \tag{9.9}$$

For any link weight L, where k denotes the k'th input word considered and where $F(k)$ is a small increment that varies over time (over k).

In certain realizations of the LAMSTAR, the forgetting adjustment is set as:

$$F(k) = 0 \quad \text{over successive } p-1 \text{ input words considered;} \tag{9.10-a}$$

but

$$F(k) = bL(k) \quad \text{per each } p\text{'th input word} \tag{9.10-b}$$

where L is any link weight and

$$b < 1 \tag{9.10-c}$$

say, $b = 0.5$.

Furthermore, in preferred realizations $L_{\max}$ is unbounded, except for reductions due to forgetting.

Noting the forgetting formula of Eqs. (9.9) and (9.10), link weights $L_{i,j}$ decay over time. Hence, if not chosen successfully, the appropriate $L_{i,j}$ will drop towards zero. Therefore, correlation links L which do not participate in successful diagnosis/decision over time, or lead to an incorrect diagnosis/decision are gradually forgotten. The forgetting feature allows the network to rapidly retrieve very recent information. Since the value of these links decreases only gradually and does not drop immediately to zero, the network can re-retrieve information associated with those links. The forgetting feature of the LAMSTAR network helps to avoid the need to consider a very large number of links, thus contributing to the network efficiency. At the forgetting feature requires storage of link weights and numbering of input words. Hence, in the simplest application of forgetting, old link weights are forgotten (subtracted from their current value) after, say every M input words. The forgetting can be applied gradually rather than stepwise as in Eqs. (9.5) above.

A stepwise Forgetting algorithm can be implemented such that all weights and decisions must have an index number k ($k = 1, 2, 3, \ldots$), starting from the very first entry. Also, then one must remember the weights as they are every M (say, $M = 20$) input words. Consequently, one updates ALL weights every $M = 20$ input words by subtracting from EACH weight its stored value to be forgotten.

For example, at input word $k = 100$ one subtracts the weights as of Input Word $k = 20$ (or alternatively X%, say, 50% thereof) from the corresponding weights at input word $k = 100$ and thus one KEEPS only the weights of the last 80 input words. Updating of weights is otherwise still done as before and so is the advancement of k. Again, at input word $k = 120$ one subtracts the weights as of input word $k = 40$ to keep the weights for an input-words interval of duration of, say, $P = 80$, and so on. Therefore, at $k = 121$ the weights (after the subtraction above) cover experience relating to a period of 81 input words. At $k = 122$, they cover a stored-weights experience over 82 input words $\ldots$, at $k = 139$ they covers a period of 99 input words, at $k = 140$ they cover 120–20 input words, since now one subtracted the weights of $k = 40$, etc. Hence, weights cover always a period of no more than 99 input words and no less than 80 input words. Weights must then be stored only every $M = 20$ input words, not per every input word. Note that the $M = 20$ and $P = 80$ input words mentioned are arbitrary. When one wishes to keep data over longer periods, one may set M and P to other values as desired.

Simple applications of the LAMSTAR neural network do not always require the implementation of the forgetting feature. If in doubt about using the forgetting property, it may be advisable to compare performance "***with forgetting***" against "***without forgetting***" (when continuing the training throughout the testing period).

9.5. Training vs. Operational Runs

There is no reason to stop training as the first n sets (input words) of data are only to establish initial weights for the testing set of input words (which are, indeed, normal run situations), which, in LAMSTAR, we can still continue training set by set (input-word by input-word). Thus, the NETWORK continues adapting itself during testing and regular operational runs. The network's performance benefits significantly from continued training while the network does not slow down and no additional complexity is involved. In fact, this does slightly simplify the network's design. Still, in scoring network performance, a number of initial runs should not be considered, since the network has not "learnt" sufficiently and is too far from convergence. However, If decision is still needed early on, then even "untrained" outputs can be used, though at at the risk of being wrong".

9.5.1. *INPUT WORD for training and for information retrieval*

In applications such as medical diagnosis, the LAMSTAR system is trained by entering the symptoms/diagnosis pairs (or diagnosis/medication pairs). The *training* input word **X** is then of the following form:

$$\underline{X} = [\underline{x}_1^T, \underline{x}_2^T, \ldots, \underline{x}_n^T, \underline{d}_1^T, \ldots, \underline{d}_k^T]^T \tag{9.11}$$

where $\boldsymbol{x}_i$ are input subwords and $\boldsymbol{d}_i$ are subwords representing past outputs of the network (diagnosis/decision). Note also that one or more SOM modules may serve as output modules to output the LAMSTAR's decisions/diagnoses.

The input word of Eqs. (9.2) and (9.11) is set to be a set of coded subword (Sec. 9.2), comprising of coded vector-subwords ($\boldsymbol{x}_i$) that relate to various categories (input dimensions). Also, each SOM module of the LAMSTAR network corresponds to one of the categories of $\underline{x}_i$ such that the number of SOM modules equals the number of sub-vectors (subwords) $\boldsymbol{x}_n$ and $\boldsymbol{d}$ in $\boldsymbol{X}$ defined by Eq. (9.11).

9.6. Operation in Face of Missing Data

The decision Eq. (9.7) of Sec. 9.3.5 above is fully applicable even when some data subwords are missing from any given input word, since the summation over k is still valid when some k are missing. In that case, the summation over k just ignores some values just as a physician can make a diagnostic decision, if need be, even when one result item did not come back from the lab. The LAMSTAR ANN is therefore fully operational in case of missing data or data set. In that case, of course, the decision may not be as good as when all subwords were available, just as the physician's decision when one or a few lab tests are missing. But, if decision must be made (say, to save a critical patient) the doctor may still go ahead with the best assessment of the information that is available.

9.7. Advanced Data Analysis Capabilities

Since all information in the LAMSTAR network is encoded in the correlation links, the LAMSTAR can be utilized as a data analysis tool. In this case the system provides analysis of input data such as evaluating the importance of input subwords, the strengths of correlation between categories, or the strengths of correlation of between individual neurons.

The system's analysis of the input data involves two phases:

(1) training of the system (as outlined in Sec. 9.5)
(2) analysis of the values of correlation links as discussed below.

Since the correlation links connecting clusters (patterns) among categories are modified (increased/decreased) in the training phase, it is possible to single out the links with the highest values. Therefore, the clusters connected by the links with the highest values determine the trends in the input data. In contrast to using data averaging methods, isolated cases of the input data will not affect the LAMSTAR results, noting its forgetting feature. Furthermore, the LAMSTAR structure makes it very robust to missing input subwords.

After the training phase is completed, the LAMSTAR system finds the highest correlation links (link weights) and reports messages associated with the clusters in SOM modules connected by these links. The links can be chosen by two methods: (1) links with value exceeding a pre-defined threshold, (2) a pre-defined number of links with the highest value.

9.7.1. *Feature extraction and reduction in the LAMSTAR NN*

Features can be extracted and reduced in the LAMSTAR network according to the derivations leading to the properties of certain elements of the LAMSTAR network as follows:

Definition I: A feature can be extracted by the matrix $A(i,j)$ where i denotes a winning neuron in SOM storage module j. All winning entries are 1 while the rest are 0. Furthermore, $A(i,j)$ can be reduced via considering properties (b) to (e) below.

(a) The ***most (least) significant subword*** (winning memory neuron) {i} ***over all SOM modules*** (i.e., over the whole NN) ***with respect to a given output decision*** $\{dk\}$ ***and over all input words***, denoted as $[i^*, s^*/dk]$, is given by:

$$[i^*, s^*/dk] : L(i, s/dk) \geq L(j, p/dk) \text{ for any winning neuron } \{j\} \text{ in any module } \{\text{p}\} \tag{9.12}$$

where p is not equal to s, $L(j, p/dk)$ denoting the link weight between the j'th (winning) neuron in layer p and the winning output-layer neuron dk. Note that for determining the least significant neuron, the inequality as above is reversed.

(b) The ***most (least) significant SOM module*** $\{s^{**}\}$ ***per a given winning output decision*** $\{dk\}$ ***over all input words***, is given by:

$$s^{**}(dk) : \sum_i(\{L(i, s/dk)\} \geqq \sum_j(\{L(j, p/dk)\} \quad \text{for any module } p \tag{9.13}$$

Note that for determining the least significant module, the inequality above is reversed.

(c) The neuron $\{i^{**}(dk)\}$ that is ***most (least) significant in a particular SOM module*** (s) ***per a given output decision*** (dk)***, over all input words per a given class of problems***, is given by $i^*(s, dk)$ such that:

$$L(i, s/dk) \geqq L(j, s/dk) \quad \text{for any neuron } (j) \text{ in same module } (s)\,. \tag{9.14}$$

Note that for determining the least significant neuron in module (s), the inequality above is reversed.

(d) **Definition II:** ***REDUNDANCY:*** If **whenever** a particular neuron (i) in SOM input layer (s) is the winner for any input word considered by the LAMSTAR (for a given class of problems assigned to it) with respect to decision dk, **then also** neuron (j) in layer (t) is a winner for its particular subword of the same input word, **and when** such unique pairing holds for all and every neurons in both layers (s) and (t), then one of these two layers (s and t) is REDUNDANT.

Definition III: If the number of $\{q(p)\}$ neurons is less than the number of $\{p\}$ neurons, then layer $\{b\}$ is called an ***INFERIOR LAYER*** to $\{a\}$.

Also see Property (h) below on redundancy determination via correlation-layers.

(e) **Definition IV:** ***ZERO-INFORMATION REDUNDANCY:*** If only one neuron is ALWAYS the winner in layer (k), regardless of the output decision, then the layer contains no information and is redundant.

The above definitions and properties can serve to reduce number of features or memories by considering only a reduced number of most-significant modules or memories or by eliminating the least significant ones.

9.7.2. *Correlation, Interpolation, Extrapolation and Innovation-Detection*

(f) ***Correlation feature***

Consider the (m) most significant layers (modules) with respect to output decision (dk) and the (n) most significant neurons in each of these (m) layers, with respect to the same output decision. (Example: Let $m = n = 4$). We comment that correlation between subwords can also be accommodated in the network by assigning a specific input subword of that correlation, this subword being formed by pre-processing.

Correlation-Layer Set-Up Rule: Establish additional SOM layers denoted as CORRELATION-LAYERS $\lambda(p/q, dk)$, such that the number of these additional correlation-layers is:

$$\sum_{i=1}^{m-1} i_{(\text{per output decision } dk)} \tag{9.15}$$

(Example: The correlation-layers for the case of $n = m = 4$ are: $\lambda(1/2, dk)$; $\lambda(1/3, dk)$; $\lambda(1/4, dk)$; $\lambda(2/3, dk)$; $\lambda(2/4, dk)$; $\lambda(3/4, dk)$.)

Subsequently, WHENEVER neurons $N(i,p)$ and $N(j,q)$ are simultaneously (namely, for the same given input word) winners at layers (p) and (q) respectively, and both these neurons also belong to the subset of 'most significant' neurons in 'most significant' layers (such that p and q are 'most significant' layers), THEN we declare a neuron $N(i,p/j,q)$ in Correlation-Layer $\lambda(p/q, dk)$ to be the winning neuron in that correlation-layer and we reward/punish its output link-weight $L(i,p/j,q-dk)$ as need be for any winning neuron in any other input SOM layer.

(Example: The neurons in correlation-layer $\lambda(p/q)$ are: $N(1,p/1,q)$; $N(1,p/2,q)$; $N(1,p/3,q)$; $N(1,p/4,q)$, $N(2,p/1,q)$; $\ldots N(2,p/4,q)$; $N(3,p/1,q)$; $\ldots N(4,p/1,q)$; $\ldots N(4,p/4,q)$, to total mxm neurons in the correlation-layer).

Any winning neuron in a correlation layer is treated and weighted as any winning neuron in another (input-SOM) layer as far as its weights to any output layer neuron are concerned and updated. Obviously, a winning neuron (per a given input word), if any, in a correlation layer p/q is a neuron $N(i,p/j,q)$ in that layer where both neuron $N(i,p)$ in input layer (p) and neuron $N(j,q)$ in layer (q) were winners for the given input word.

(g) ***Interpolation/Extrapolation via Correlation Layers:*** Let p be a 'most significant' layer and let i be a 'most significant neuron with respect to output decision dk in layer p, where no input subword exists in a given input word relating to layer p. Thus, neuron $N(i,p)$ is considered as the interpolation/extrapolation neuron for layer p if it satisfies:

$$\sum_q \{L(i,p/w,q-dk)\} \geqq \sum_q \{L(v,p/w,q-dk)\} \tag{9.16}$$

where v are different from i and where $L(i,p/j,q \to dk)$ denote link weights from correlation-layer $\lambda(p/q)$. Note that in every layer q there is only one winning neuron for the given input word, denoted as $N(w,q)$, whichever w may be at any q'th, layer.

(Example: Let $p = 3$. Thus consider correlation-layers $\lambda(1/3, dk)$; $\lambda(2/3, d\kappa)$; $\lambda(3/4, dk)$ such that: $q = 1, 2, 4$.) Obviously, no punishment/reward is applied to a neuron that is considered to be the interpolation/extrapolation of another neuron not actually arising from the input word itself.

(h) ***Redundancy via Correlation-Layers:*** Let p be a 'most significant' layer and let i be a 'most significant' neuron in that layer. Layer p is redundant if for all input words there is there is another 'most significant' layer q such that, for any output decision and for any neuron $N(i, p)$, only one correlation neuron $i, p/j, q$ (i.e., for only one j per each such i, p) has non-zero output-link weights to any output decision dk, such that every neuron $N(j, p)$ is always associated with only one neuron $N(j, p)$ in some layer p.

(Example: Neuron $N(1, p)$ is always associated with neuron $N(3, q)$ and never with $N(1, q)$ or $N(2, q)$ or $N(4, q)$, while neuron $N(2, p)$ is always associated with $N(4, q)$ and never with other neurons in layer q).

Also, see property (d) above.

9.7.3. *Innovation detection in the LAMSTAR NN*

(i) If link-weights from a given input SOM layer to the output layer output change considerably and repeatedly (beyond a threshold level) within a certain time interval (a certain specified number of successive input words that are being applied), relatively to link weights from other input SOM layers, then ***innovation*** is detected with respect to that input layer (category).

(j) ***Innovation*** is also detected if weights between neurons from one input SOM layer to another input SOM layer similarly change.

9.8. Modified Version: Normalized Weights

A somewhat modified version of the LAMSTAR is proposed in (Sneider, Graupe, 2008), where the link weights $L_{i,j}(m, k)$ from neuron m in the k'th SOM input layer to any output layer j in the i'th output (decision) layer is replaced by a normalized link weight denoted as $L^*_{i,j}(m, k)$ where

$$L^*_{i,j}(m, k) = L_{i,j}(m, k)/n(m, k) \tag{9.17}$$

$n(m, k)$ denoting the count of the number of times when neuron m in input layer k is the winning input neuron in that layer.

Consequently, the winning decision, as in Eq. (9.7) will employ L^* rather than L throughout. Similarly L^* will replace L in weight links between any two different input layers, if applicable.

This modification is important when certain input neurons are significant even though the occur (become "winners") only rarely. It proved important in several applications, such as (Waxman *et al.*, 2010), where it greatly outperformed the un-normalized version of the LAMSTAR network.

9.9. Concluding Comments and Discussion of Applicability

The LAMSTAR neural network utilizes the basic features of many other neural network, and adopts Kohonen's SOM modules [Kohonen, 1977, 1984] with their associative-memory — based setting of storage weights (w_{ij} in this Chapter) and its WTA (Winner-Take-All) feature, it differs in its neuronal structure in that every neuron has not only storage weights w_{ij} (see Chap. 8 above), but also the link weights L_{ij}. This feature directly follows Hebb's Law [Hebb, 1949] and its relation to Pavlov's Dog experiment, as discussed in Sec. 3.1. It also follows Minsky's k-lines model [Minsky, 1980] and Kant's emphasis [Kant, 1781] on the essential role Verbindungen in "understanding". Hence, not only does LAMSTAR deal with two kinds of neuronal weights (for storage and for linkage to other layers), but in the LAMSTAR, the link weights are the ones that count for decision purposes. The storage weights form "atoms of memory" in the Kantian sense [Ewing, 1938]. The LAMSTAR's decisions are solely based on these link weights — see Sec. 9.3 below.

The LAMSTAR, like most neural networks, attempts to provide a representation of the problem it must solve (Rosenblatt, 1961). This representation, regarding the networks decision, can be formulated in terms of a nonlinear mapping **L** of the weights between the inputs (input vector) and the outputs, that is arranged in a matrix form. Therefore, **L** is a nonlinear mapping function whose entries are the weights between inputs and the outputs, which map the inputs to the output decision. Considering the Back-Propagation (BP) network, the weights in each layer are the columns of **L**. The same holds for the link weights L_{ij} of **L** to a winning output decision in the LAMSTAR network. Obviously, in both BP and LAMSTAR, **L** is not a square matrix-like function, nor are all its columns of same length. However, in BP, **L** has ***many entries*** (weights) in ***each*** column per any output decision. In contrast, in the LAMSTAR, each column of **L** has ***only one non-zero entry***. This accounts both for the **speed** and the **transparency** of LAMSTAR. There weights in BP do not yield direct information on what their values mean. In the LAMSTAR, the link weights directly indicate the significance of a given feature and of a particular subword relative to the particular decision, as indicated in Sec. 9.5 below. The basic LAMSTAR algorithm requires the computation of only Eqs. (9.5) and (9.7) per iteration. These usually involve only addition/subtraction and thresholding operations while no multiplication is involved, to further contribute to the LAMSTAR's computational speed.

The LAMSTAR network facilitates a multidimensional analysis of input variables to assign, for example, different weights (importance) to the items of data, find correlation among input variables, or perform identification, recognition and clustering of patterns. Being a neural network, the LAMSTAR can do all this without re-programming for each diagnostic problem.

The decisions of the LAMSTAR neural network are based on many categories of data, where often some categories are fuzzy while some are exact, and often categories are missing (incomplete data sets). As mentioned in Sec. 9.1 above, the LAMSTAR network can be trained with incomplete data or category sets. Therefore, due to its features, the LAMSTAR neural network is a very effective tool in just such situations. As an input, the system accepts data defined by the user, such as, system state, system parameters, or very specific data as it is shown in the application examples presented below. Then, the system builds a model (based on data from past experience and training) and searches the stored knowledge to find the best approximation/description to the features/parameters given as input data. The input data could be automatically sent through an interface to the LAMSTAR's input from sensors in the system to be diagnosed, say, an aircraft into which the network is built in.

The LAMSTAR system can be utilized as:

— Computer-based medical diagnosis system [Kordylewski and Graupe, 2001, Nigam and Graupe, 2004, Muralidharan and Rousche, 2005, Waxman *et al.*, 2010].
— Tool for financial evaluations.
— Tool for industrial maintenance and fault diagnosis (on same lines as applications to medical diagnosis).
— Tool for data mining [Carino *et al.*, 2005] and financial decisions (Sec. 9.C below).
— Tool for browsing and information retrieval.
— Tool for data analysis, classification, browsing, and prediction [Sivaramakrishnan and Graupe, 2004], Case Studies 9.B, 9.C, 9.D.
— Tool for image detection and recognition, See: Sec. 9.D, and [Girado *et al.*, 2004].
— Teaching aid.
— Tool for analyzing surveys and questionnaires on diverse items.

All these applications can employ many of the other neural networks that we discussed. However, the LAMSTAR has certain advantages, such as insensitivity to initialization, the avoidance of local minima, its forgetting capability (this can often be implemented in other networks), its transparency (the link weights carry clear information as to the link weights on relative importance of certain inputs, on their correlation with other inputs, on innovation detection capability and on redundancy of data — see Secs. 9.7 above). The latter allow downloading data without prior determination of its significance and letting the network decide for itself, via the link weights to the outputs. The LAMSTAR, in contrast to many other networks, can work uninterrupted if certain sets of data (input-words) are incomplete (missing subwords) without requiring any new training or algorithmic changes. Similarly, input subwords can be added during the network's operation without reprogramming while taking advantage of its forgetting feature. Furthermore, the LAMSTAR is very fast, especially in comparison to back-propagation or to statistical networks, without sacrificing performance and it always learns during regular runs.

Appendix 9.A provides details of the LAMSTAR algorithm for the Character Recognition problem that was also the subject of Appendices to Chaps. 5, 6, 7, 8, 10, 12 and 13. Several different examples of applications to medical decision and diagnosis problems are given in Appendix 9.B below.

Case study 9.C is a financial application, which also compares performance of the LAMSTAR with Back Propagation (including its RBF (Radial Basis Function) version and SVM (Support Vector Machine), which lies outside thefield of neural networks, for the same problem. Appendix 9.D describes an application to astronomy, for recognizing constellations.

9.A. LAMSTAR Network Case Study*: Character Recognition

9.A.1. *Introduction*

This case study focuses on recognizing characters '6', '7', 'X' and "rest of the world" patterns namely, patterns not belonging to the set '6', '7', 'X'). The characters in the training and testing set are represented as unipolar inputs '1' and '0' in a $6 * 6$ grid. An example of a character is as follows:

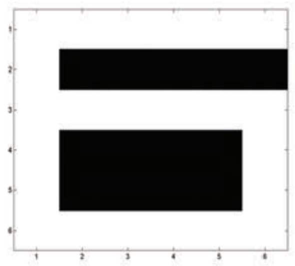


Fig. 9.A.1. Example of a training pattern ('6').

1	1	1	1	1	1
1	0	0	0	0	0
1	1	1	1	1	1
1	0	0	0	0	1
1	0	0	0	0	1
1	1	1	1	1	1

Fig. 9.A.2. Unipolar Representation of '6'.

*Computed by Vasanth Arunachalam, ECE Dept., University of Illinois, Chicago, 2005.

9.A.2. *Design of the network*

The LAMSTAR network has the following components:

(a) *INPUT WORD AND ITS SUBWORDS:*

The input word (in this case, the character) is divided into a number of subwords. Each subword represents an attribute of the input word. The subword division in the character recognition problem was done by considering every row and every column as a subword hence resulting in a total of 12 subwords for a given character.

(b) *SOM MODULES FOR STORING INPUT SUBWORDS:*

For every subword there is an associated Self Organizing Map (SOM) module with neurons that are designed to function as Kohonen 'Winner Take All' neurons where the winning neuron has an output of 1 while all other neurons in that SOM module have a zero output.

In this project, the SOM modules are built dynamically in the sense that instead of setting the number of neurons at some fixed value arbitrarily, the network was built to have neurons depending on the class to which a given input to a particular subword might belong. For example if there are two subwords that have all their pixels as '1's, then these would fire the same neuron in their SOM layer and hence all they need is 1 neuron in the place of 2 neurons. This way the network is designed with lesser number of neurons and the time taken to fire a particular neuron at the classification stage is reduced considerably.

(c) *OUTPUT (DECISION) LAYER:*

The present output layer is designed to have two layers, which have the following neuron firing patterns:

Table 9.A.1. Firing order of the output neurons.

Pattern	Output Neuron 1	Output Neuron 2
'6'	Not fired	Not fired
'7'	Not fired	Fired
'X'	Fired	Not fired
'Rest of the World'	Fired	Fired

The link-weights from the input SOM modules to the output decision layer are adjusted during training on a reward/punishment principle. Furthermore, they continue being trained during normal operational runs. Specifically, if the output of the particular output neuron is what is desired, then the link weights to that neuron is rewarded by increasing it by a non-zero increment, while punishing it by a small non-zero number if the output is not what is desired.

Note: The same can be done (correlation weights) between the winning neurons of the different SOM modules but has not been adopted here due to the complexities involved in implementing the same for a generic character recognition problem.

The design of the network is illustrated in Fig. 9.A.3.

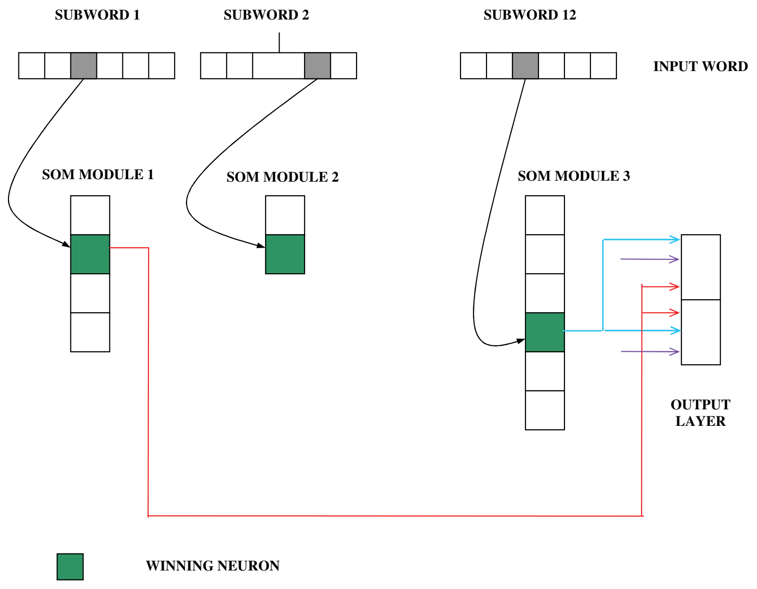


Fig. 9.A.3. Design of the LAMSTAR neural network for character recognition. Number of SOM modules in the network is 12. The neurons (Kohonen) are designed to build dynamically which enables an adaptive design of the network. Number of neurons in the output layer is 2. There are 12 subwords for every character input to the network. Green denotes the winning neuron in every SOM module for the respective shaded subword pixel. Reward/Punishment principle is used for the output weights.

9.A.3. *Fundamental principles*

Fundamental principles used in dynamic SOM layer design

As explained earlier the number of neurons in every SOM module is not fixed. The network is designed to grow dynamically. At the beginning there are no neurons in any of the modules. So when the training character is sent to the network, the first neuron in every subword is built. Its output is made 1 by adjusting the weights based on the 'Winner Take All' principle. When the second training pattern is input to the system, this is given as input to the first neuron and if the output is close to 1 (with a tolerance value of 0.05), then the same neuron is fired and another neuron is not built. The second neuron is built only when a distinct subword appears at the input of all the previously built neuron resulting in their output not being sufficiently close to 1 so as to declare any of them a winning neuron.

It has been observed that there has been a significant reduction in the number of neurons required in every SOM modules.

Winner Take All principle

The SOM modules are designed to be Kohonen layer neurons, which act in accordance to the 'Winner Take All' Principle. This layer is a competitive layer wherein the Eucledian distance between the weights at every Kohonen layer and the input pattern is measured and the neuron that has the least distance if declared to be the winner. This Kohonen neuron best represents the input and hence its output is made equal to 1 whereas all other neuron outputs are forced to go to 0. This principle is called the 'Winner Take All' principle. During training the weights corresponding to the winning neuron is adjusted such that it closely resembles the input pattern while all other neurons move away from the input pattern.

9.A.4. *Training algorithm*

The training of the LAMSTAR network if performed as follows:

(i) **Subword Formation:**
The input patterns are to be divided into subwords before training/testing the LAMSTAR network. In order to perform this, the every row of the input 6*6 character is read to make 6 subwords followed by every column to make another 6 subwords resulting in a total of 12 subwords.

(ii) **Input Normalization:**
Each subwords of every input pattern is normalized as follows:

$$xi' = x_i \Big/ \sqrt{\Sigma x_j^2}$$

where, x — subword of an input pattern. During the process, those subwords, which are all zeros, are identified and their normalized values are manually set to zero.

(iii) **Rest of the world Patterns:**
The network is also trained with the rest of the world patterns 'C', 'I' and '$\|$'. This is done by taking the average of these patterns and including the average as one of the training patterns.

(iv) **Dynamic Neuron formation in the SOM modules:**
The first neuron in all the SOM modules are constructed as Kohonen neurons as follows:

- As the first pattern is input to the system, one neuron is built with 6 inputs and random weights to start with initially and they are also normalized just like the input subwords. Then the weights are adjusted such that the output of this neuron is made equal to 1 (with a tolerance of 10^{-5} according to the formula:

$$w(n+1) = w(n) + \alpha^*(x - w(n))$$

where,
α — learning constant = 0.8
w — weight at the input of the neuron
x — subword

$$z = w^* x$$

where, z — output of the neuron (in the case of the first neuron it is made equal to 1).

- When the subwords of the subsequent patterns is input to the respective modules, the output at any of the previously built neuron is checked to see if it is close to 1 (with a tolerance of 0.05). If one of the neurons satisfies the condition, then this is declared as the winning neuron, i.e., a neuron whose weights closely resemble the input pattern. Else another neuron is built with new sets of weights that are normalized and adjusted as above to resemble the input subword.
- During this process, if there is a subword with all zeros then this will not contribute to a change in the output and hence the output is made to zero and the process of finding a winning neuron is bypassed for such a case.

(v) **Desired neuron firing pattern:**
The output neuron firing pattern for each character in the training set has been established as given in Table 1.

(vi) **Link weights:**
Link weights are defined as the weights that come from the winning neuron at every module to the 2 output neurons. If in the desired firing, a neuron is to be fired, then its corresponding link weights are rewarded by adding a small positive value of 0.05 every iteration for 20 iterations. On the other hand, if a neuron should not be fired then its link weights are reduced 20 times by 0.05. This will result in the summed link weights at the output layer being a positive value indicating a fired neuron if the neuron has to be fired for the pattern and high negative value if it should not be fired.

(vii) The weights at the SOM neuron modules and the link weights are stored.

9.A.4.1. *Training set*

The LAMSTAR network is trained to detect the characters ‘6’, ‘7’, ‘X’ and ‘rest of the world’ characters. The training set consists of 16 training patterns 5 each for ‘6’, ‘7’ and ‘X’ and one average of the ‘rest of the world’ characters.

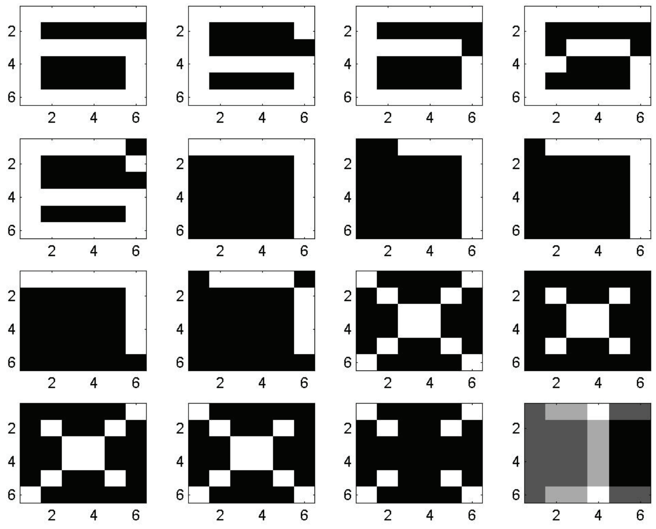


Fig. 9.A.4. Training Pattern Set for recognizing characters ‘6’, ‘7’, ‘X’ and ‘mean of rest of world’ patterns ‘C’, ‘I’, ‘||’.

9.A.4.2. *‘Rest of the world’ patterns*

The rest of the world patterns used to train the network are as follows:

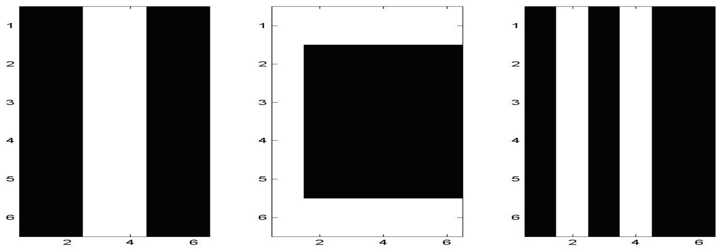


Fig. 9.A.5. ‘Rest of the world patterns ‘I’, ‘C’ and ‘||’.

9.A.5. ***Testing procedure***

The LAMSTAR network was tested with 8 patterns as follows:

- The patterns are processed to get 12 subwords as before. Normalization is done for the subwords as explained in the training.
- The stored weights are loaded
- The subwords are propagated through the network and the neuron with the maximum output at the Kohonen layer is found and their link weights are sent to the output neurons.

- The output is a sum of all the link weights.
- All the patterns were successfully classified. There were subwords that were completely zero so that the pattern would be partially incorrect. Even these were correctly classified.

9.A.5.1. *Test pattern set*

The network was tested with 8 characters consisting of 2 pattern each of '6', '7', 'X' and rest of the world. All the patterns are noisy, either distorted or a whole row/column removed to test the efficiency of the training. The following is the test pattern set.

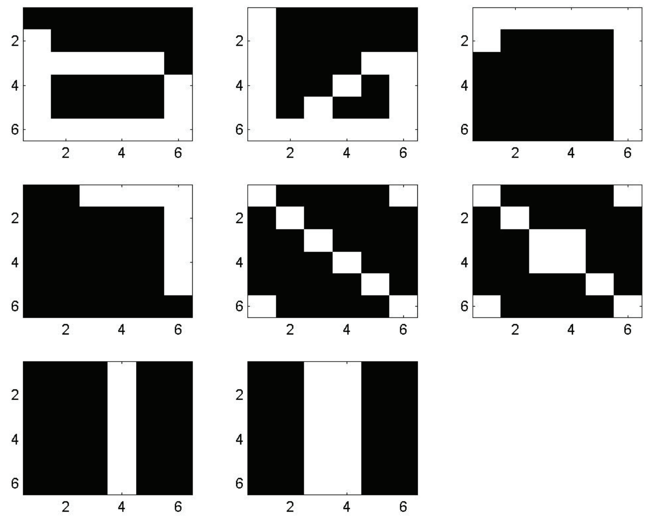


Fig. 9.A.6. Test pattern set consisting of 2 patterns each for '2', '7', 'X' and 'rest of the world'.

9.A.6. ***Results and their analysis***

9.A.6.1. *Training results*

The results obtained after training the network are presented in Table 9.A.2:

- Number of training patterns = 16
- Training efficiency = 100%
- Number of SOM modules = 12
- The number of neurons in the 12 SOM modules after dynamic neuron formation in are:

Table 9.A.2. Number of neurons in the SOM modules.

SOM Module Number	Number of neurons
1	3
2	2
3	2
4	4
5	2
6	4
7	3
8	3
9	3
10	3
11	3
12	7

9.A.6.2. *Test results*

The result of testing the network are as in Table 9.A.3:

- Number of testing patterns = 8
- Neurons fired at the modules for the 8 test patterns:

Table 9.A.3. Neurons fired during the testing for respective patterns.

	Module Number											
Pattern	1	2	3	4	5	6	7	8	9	10	11	12
6	0	0	0	1	1	1	1	1	1	1	1	4
6	0	0	1	1	1	1	1	2	3	2	1	1
7	1	1	2	4	2	2	1	2	3	2	1	5
7	1	2	2	4	2	2	1	2	3	2	1	5
X	2	2	2	4	2	3	2	2	3	2	1	6
X	2	2	2	4	2	3	2	2	3	2	1	6
|	2	2	2	4	2	3	2	2	3	3	1	6
||	2	2	2	4	2	3	2	2	3	3	1	6

The firing pattern of the output neurons for the test set is given in Table 9.A.4:

- Efficiency: 100%.

Table 9.A.4. Firing pattern for the test characters.

Test Pattern	Neuron 1	Neuron 2
6 (with bit error)	−25.49 (Not fired)	−25.49 (Not fired)
6 (with bit error)	−20.94 (Not fired)	−20.94 (Not fired)
7 (with bit error)	−29.99 (Not fired)	15.99 (Fired)
7 (with bit error)	−24.89 (Not fired)	18.36 (Fired)
X (with bit error)	9.99 (Fired)	−7.99 (Not fired)
X (with bit error)	9.99 (Fired)	−7.99 (Not fired)
| (with bit error)	0.98 (Fired)	0.98 (Fired)
|| (with bit error)	1.92 (Fired)	1.92 (Fired)

9.A.7. *Summary and concluding observations*

Summary:

- Number of training patterns = 16 (5 each of '6', '7', 'X' and 1 mean image of 'rest of the world'
- Number of test patterns = 8 (2 each for '6', '7', 'X' and 'rest of the world' with bit errors)
- Number of SOM modules = 12
- Number of neurons in the output layer = 2
- Number of neurons in the SOM module changes dynamically. Refer table 2 for the number of neurons in each module.
- Efficiency = 100%

Observations:

- The network was much faster than the Back Propagation network for the same character recognition problem.
- By dynamically building the neurons in the SOM modules, the number of computations is largely reduced as the search time to find the winning neuron is reduced to a small number of neurons in many cases.
- Even in the case when neurons are lost (simulated as a case where the output of the neuron is zero i.e., all its inputs are zeros), the recognition efficiency is 100%. This is attributed to the link weights, which takes cares of the above situations.
- The NN learns as it goes even if untrained
- The test patterns where all noisy (even at several bits, yet efficiency was 100%.

9.A.8. *LAMSTAR SOURCE CODE* (*MATLAB*)

Main.m

```
clear all
close all

X = train_pattern;
%pause(1)
%close all

n = 12 % Number of subwords
flag = zeros(1,n);

% To make 12 subwords from 1 input
for i = 1:min(size(X)),
   X_r{i} = reshape(X(:,i),6,6);
   for j = 1:n,
   if (j<=6),
       X_in{i}(j,:) = X_r{i}(:,j)';
   else
       X_in{i}(j,:) = X_r{i}(j-6,:);
   end
end

% To check if a subword is all '0's and makes it normalized value equal to zero
% and to normalize all other input subwords
 p(1,:) = zeros(1,6);
 for k = 1:n,
    for t = 1:6,
       if (X_in{i}(k,t)~= p(1,t)),
           X_norm{i}(k,:) = X_in{i}(k,:)/sqrt(sum(X_in{i}(k,:).^2));
       else
           X_norm{i}(k,:) = zeros(1,6);
       end
    end
 end
 end%%%End of for

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Dynamic Building of neurons
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 % Building of the first neuron is done as Kohonen Layer neuron
 %(this is for all the subwords in the first input pattern for all SOM modules

 i = 1;
 ct = 1;
 while (i<=n),
  i
  cl = 0;
  for t = 1:6,
     if (X_norm{ct}(i,t)==0),
         cl = cl+1;
     end
  end
```

```
if (cl == 6),
    Z{ct}(i) = 0;
elseif (flag(i) == 0),
   W{i}(:,ct) = rand(6,1);
   flag(i) = ct;
   W_norm{i}(:,ct) = W{i}(:,ct)/sqrt(sum(W{i}(:,ct).^2));
   Z{ct}(i)= X_norm{ct}(i,:)*W_norm{i};

   alpha =0.8;
   tol = 1e-5;

   while(Z{ct}(i) <= (1-tol)),
     W_norm{i}(:,ct) = W_norm{i}(:,ct) + alpha*(X_norm{ct}(i,:)' -
W_norm{i}(:,ct));
     Z{ct}(i) =  X_norm{ct}(i,:)*W_norm{i}(:,ct);
   end%%%%End of while
  end%%%End of if
  r(ct,i) = 1;
  i = i+1;
end%%%End of while


r(ct,:) = 1;
ct = ct+1;
while (ct <= min(size(X))),
  for i = 1:n,
     cl = 0;
     for t = 1:6,
        if (X_norm{ct}(i,t)==0),
            cl = cl+1;
        end
     end

     if (cl == 6),
         Z{ct}(i) = 0;
     else
        i
        r(ct,i) = flag(i);
        r_new=0;
        for k = 1:max(r(ct,i)),
           Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,k);

           if Z{ct}(i)>=0.95,
              r_new = k;
              flag(i) = r_new;
              r(ct,i) = flag(i);
              break;
           end%%End of if
        end%%%%%%End of for

        if (r_new==0),
            flag(i) = flag(i)+1;
            r(ct,i) = flag(i);
            W{i}(:,r(ct,i)) = rand(6,1);
            %flag(i) = r
```

```
          W_norm{i}(:,r(ct,i)) = W{i}(:,r(ct,i))/sqrt(sum(W{i}(:,r(ct,i)).^2));
          Z{ct}(i) = X_norm{ct}(i,:)*W_norm{i}(:,r(ct,i));

          alpha =0.8;
          tol = 1e-5;

          while(Z{ct}(i) <= (1-tol)),
            W_norm{i}(:,r(ct,i)) = W_norm{i}(:,r(ct,i)) +
  alpha*(X_norm{ct}(i,:)' -
  W_norm{i}(:,r(ct,i)));
            Z{ct}(i) =  X_norm{ct}(i,:)*W_norm{i}(:,r(ct,i));
          end%%%End of while
       end%%%End of if
       %r_new
       %disp('Flag')
       %flag(i)
     end%%%%End of if
  end

  ct = ct+1;
end
save W_norm W_norm

for i = 1:5,
    d(i,:) = [0 0];
    d(i+5,:) = [0 1];
    d(i+10,:) = [1 0];
end
d(16,:) = [1 1];
%%%%%%%%%%%%%%%
% Link Weights
%%%%%%%%%%%%%%%
ct = 1;
m_r = max(r);
for i = 1:n,
   L_w{i} = zeros(m_r(i),2);
   end
ct = 1;

%%% Link weights and output calculations
Z_out = zeros(16,2);
while (ct <= 16),
  ct
  %for mn = 1:2
  L = zeros(12,2);
  %    for count = 1:20,
  for i = 1:n,
    if (r(ct,i)~=0),
       for j = 1:2,
           if (d(ct,j)==0),
               L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)-0.05*20;
           else
               L_w{i}(r(ct,i),j) = L_w{i}(r(ct,i),j)+0.05*20;
           end %%End if loop
       end %%% End for loop
```

```
            L(i,:) = L_w{i}(r(ct,i),:);
         end %%%End for loop
       end
       %   end %%% End for loop
       Z_out(ct,:) = sum(L);
   ct = ct+1;
   end

save L_w L_w
```

Test.m

```
clear all
X = test_pattern;
load W_norm
load L_w

% To make 12 subwords
for i = 1:min(size(X)),
   i
   X_r{i} = reshape(X(:,i),6,6);
   for j = 1:12,
       if (j<=6),
          X_in{i}(j,:) = X_r{i}(:,j)';
       else
          X_in{i}(j,:) = X_r{i}(j-6,:);
       end
   end

   p(1,:) = zeros(1,6);

   for k = 1:12,
      for t = 1:6,
         if (X_in{i}(k,t)~= p(1,t)),
             X_norm{i}(k,:) = X_in{i}(k,:)/sqrt(sum(X_in{i}(k,:).^2));
         else
             X_norm{i}(k,:) = zeros(1,6);
         end
      end
   end

   for k = 1:12,
      Z = X_norm{i}(k,:)*W_norm{k};
      if (max(Z) == 0),
          Z_out(k,:) = [0 0];
      else
        index(k) = find(Z == max(Z));
        L(k,:) = L_w{k}(index(k),:);
        Z_out(k,:) = L(k,:)*Z(index(k));
      end
   end
   final_Z = sum(Z_out)
end
```

training_pattern.m

```
function train = train_pattern

x1 = [1 1 1 1 1 1; 1 0 0 0 0 0; 1 1 1 1 1 1; 1 0 0 0 0 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];
x2 = [1 1 1 1 1 1; 1 0 0 0 0 1; 1 0 0 0 0 0; 1 1 1 1 1 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];
x3 = [1 1 1 1 1 1; 1 0 0 0 0 0; 1 1 1 1 1 0; 1 0 0 0 0 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];
x4 = [1 1 1 1 1 1; 1 0 0 0 0 0; 1 0 1 1 1 0; 1 1 0 0 0 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];
x5 = [1 1 1 1 1 0; 1 0 0 0 0 1; 1 0 0 0 0 0; 1 1 1 1 1 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];

x6 = zeros(6,6);
x6(1,:) = 1;
x6(:,6) = 1;

x7 = zeros(6,6);
x7(1,3:6) = 1;
x7(:,6) = 1;

x8 = zeros(6,6);
x8(1,2:6) = 1;
x8(:,6) = 1;

x9 = zeros(6,6);
x9(1,:) = 1;
x9(1:5,6) = 1;

x10 = zeros(6,6);
x10(1,2:5) = 1;
x10(2:5,6) = 1;

x11 = zeros(6,6);
for i = 1:6,
x11(i,i) = 1;
end
x11(1,6) = 1;
x11(2,5) = 1;
x11(3,4) = 1;
x11(4,3) = 1;
x11(5,2) = 1;
x11(6,1) = 1;

x12 = x11;

x12(1,1) = 0;
x12(6,6) = 0;
x12(1,6) = 0;
x12(6,1) = 0;

x13 = x11;
x13(1,1) = 0;
```

```
       x13(6,6) = 0;

       x14 = x11;
       x14(1,6) = 0;
       x14(6,1) = 1;

       x15 = x11;
       x15(3:4,3:4) = 0;

       x16 = zeros(6,6);
       x16(:,3:4) = 1;

       x17 = zeros(6,6);
       x17(1,:) = 1;
       x17(6,:) = 1;
       x17(:,1) = 1;

       x18 = zeros(6,6);
       x18(:,2) = 1;

       x18(:,4) = 1;

       x19 = (x16+x17+x18)/3;

       xr1 = reshape(x1',1,36);
       xr2 = reshape(x2',1,36);
       xr3 = reshape(x3',1,36);
       xr4 = reshape(x4',1,36);

       xr5 = reshape(x5',1,36);
       xr6 = reshape(x6',1,36);
       xr7 = reshape(x7',1,36);
       xr8 = reshape(x8',1,36);

       xr9 = reshape(x9',1,36);
       xr10 = reshape(x10',1,36);
       xr11 = reshape(x11',1,36);
       xr12 = reshape(x12',1,36);

       xr13 = reshape(x13',1,36);
       xr14 = reshape(x14',1,36);
       xr15 = reshape(x15',1,36);

       xr19 = reshape(x19',1,36);


       xr16 = reshape(x16',1,36);
       xr17 = reshape(x17',1,36);
       xr18 = reshape(x18',1,36);


       train = [xr1' xr2' xr3' xr4' xr5' xr6' xr7' xr8' xr9' xr10' xr11' xr12'
                xr13' xr14' xr15' xr19'];
rest = [xr16' xr17' xr18'];
```

test_pattern.m

```
function t_pat = test_pattern
x1 = [0 0 0 0 0 0; 1 0 0 0 0 0; 1 1 1 1 1 0; 1 0 0 0 0 1; 1 0 0 0 0 1;
      1 1 1 1 1 1];
x2 = zeros(6,6);
x2(:,1) = 1;
x2(3:6,6) = 1;
x2(6,:) = 1;
x2(3,5) = 1;
x2(4,4) = 1;
x2(5,3) = 1;

3 = zeros(6,6);
x3(1,:) = 1;
x3(:,6) = 1;
x3(1:2,1) = 1;

x4 = zeros(6,6);
x4(1,3:6) = 1;
x4(1:5,6) = 1;

x5 = zeros(6,6);
for i = 1:6,
   x5(i,i) = 1;
end
x5(1,6) = 1;
x5(6,1) = 1;

x6 = x5;
x6(3,4) = 1;
x6(4,3) = 1;

x7 = zeros(6,6);
x7(:,4) = 1;

x8 = zeros(6,6);
x8(:,3:4) = 1;

xr1 = reshape(x1',1,36);
xr2 = reshape(x2',1,36);
xr3 = reshape(x3',1,36);
xr4 = reshape(x4',1,36);
xr5 = reshape(x5',1,36);
xr6 = reshape(x6',1,36);
xr7 = reshape(x7',1,36);
xr8 = reshape(x8',1,36);

t_pat = [xr1' xr2' xr3' xr4' xr5' xr6' xr7' xr8'];
```

9.B. Application to Medical Diagnosis Problems

(a) Application to ESWL Medical Diagnosis Problem

In this application, the LAMSTAR network serves to aid in a typical urological diagnosis problem that is, in fact, a prediction problem [Graupe 1997, Kordylewski *et al.*, 1999]. The network evaluates a patient's condition and provides long term forecasting after removal of renal stones via Extracorporeal Shock Wave Lithotripsy (denoted as ESWL). The ESWL procedure breaks very large renal stones into small pieces that are then naturally removed from the kidney with the urine. Unfortunately, the large kidney stones appear again in 10% to 50% of patients (1–4 years post surgery). It is difficult to predict with reasonable accuracy (more than 50%) if the surgery was a success or a failure, due to the large number of analyzed variables. In this particular example, the input data (denoted as a "word" for each analyzed case, namely, for each patient) are divided into 16 subwords (categories). The length in bytes for each subword in this example varies from 1 to 6 bytes. The subwords describe patient's physical and physiological characteristics, such as patient demographics, stone's chemical composition, stone location, laboratory assays, follow-up, re-treatments, medical therapy, etc.

Table 9.B.1 below compares results for the LAMSTAR network and for a Back-Propagation (BP) neural network [Niederberger *et al.*, 1996], as applied to exactly the same training and test data sets [Kordylewski *et al.*, 1999]. While both networks model the problems with high accuracy, the results show that the LAMSTAR

Table 9.B.1. Performance comparison of the LAMSTAR network and the BP network for the renal cancer and the ESWL diagnosis.

	Renal Cancer Diagnosis		ESWL Diagnosis	
	LAMSTAR Network	BP Network Network	LAMSTAR Network	BP Network
Training Time	0.08 sec	65 sec	0.15 sec	177 sec
Test Accuracy	83.15%	89.23%	85.6%	78.79%
Negative Specificity	0.818	0.909	0.53	0.68
Positive Predictive Value	0.95	0.85	1	0.65
Negative Predictive Value	0.714	0.81	0.82	0.86
Positive Specificity	0.95	0.85	1	0.83
Wilks' Test Computation Time	< 15 mins	weeks	< 15 mins	Weeks

Comments:
Positive/Negative Predictive Values — ratio of the positive/negative cases that are correctly diagnosed to the positive/negative cases diagnosed as negative/positive.
Positive/Negative Specificity — the ratio of the positive/negative
cases that are correctly diagnosed to the negative/positive cases that are incorrectly diagnosed as positive/negative.

network is over 1000 times faster in this case. The difference in training time is due to the incorporation of an unsupervised learning scheme in the LAMSTAR network, while the BP network training is based on error minimization in a 37-dimensional space (when counting elements of subword vectors) which requires over 1000 iterations.

Both networks were used to perform the Wilks' Lambda test [Morrison, 1996, Wilks, 1938] which serves to determine which input variables are meaningful with regard to system performance. In clinical settings, the test is used to determine the importance of specific parameters in order to limit the number of patient's examination procedures.

(b) Application to Renal Cancer Diagnosis Problem

This application illustrates how the LAMSTAR serves to predict if patients will develop a metastatic disease after surgery for removal of renal-cell-tumors. The input variables were grouped into sub-words describing patient's demographics, bone metastases, histologic subtype, tumor characteristics, and tumor stage [Kordylewski *et al.*, 1999]. In this case study we used 232 data sets (patient record), 100 sets for training and 132 for testing. The performance comparison of the LAMSTAR network versus the BP network are also summarized in Table 9.B.1 above. As we observe, the LAMSTAR network is not only much faster to train (over 1000 times), but clearly gives better prediction accuracy (85% as compared to 78% for BP networks) with less sensitivity.

(c) Application to Diagnosis of Drug Abuse for Emergency Cases

In this application, the LAMSTAR network is used as a decision support system to identify the type of drug used by an unconscious patient who is brought to an emergency-room (data obtained from Maha Noujeime, University of Illinois at Chicago [Bierut *et al.*, 1998, Noujeime, 1997]). A correct and very rapid identification of the drug type, will provide the emergency room physician with the immediate treatment required under critical conditions, whereas wrong or delayed identification may prove fatal and when no time can be lost, while the patient is unconscious and cannot help in identifying the drug. The LAMSTAR system can diagnose to distinguish between five groups of drugs: alcohol, cannabis (marijuana), opiates (heroin, morphine, etc.), hallucinogens (LSD), and CNS stimulants (cocaine) [Bierut *et al.*, 1998]. In the drug abuse identification problem diagnosis can not be based on one or two symptoms since in most cases the symptoms overlap. The drug abuse identification is very complex problem since most of the drugs can cause opposite symptoms depending on additional factors like: regular/periodic use, high/low dose, time of intake [Bierut *et al.*, 1998]. The diagnosis is based on a complex relation between 21 input variables arranged in 4 categories (subword vectors) representing drug abuse symptoms. Most of these variables are easily detectable in an emergency-room setting by simple evaluation (Table 9.B.2). The large number of variables makes it often difficult for a doctor to properly interrelate them under

Table 9.B.2. Symptoms divided into four categories for drug abuse diagnosis problem.

CATEGORY 1	CATEGORY 2	CATEGORY 3	CATEGORY 4
Respiration	Pulse	Euphoria	Physical Dependence
Temperature	Appetite	Conscious Level	Psychological Dependence
Cardiac Arrhythmia	Vision	Activity Status	Duration of Action
Reflexes	Hearing	Violent Behavior	Method of Administration
Saliva Secretion	Constipation	Convulsions	Urine Drug Screen

emergency room conditions for a correct diagnosis. An incorrect diagnosis, and a subsequent incorrect treatments may be lethal to a patient. For example, while cannabis and cocaine require different treatment, when analyzing only mental state of the patient, both cannabis and large doses of cocaine can result in the same mental state classified as mild panic and paranoia. Furthermore, often not all variables can be evaluated for a given patient. In emergency-room setting it is impossible to determine all 21 symptoms, and there is no time for urine test or other drug tests.

The LAMSTAR network was trained with 300 sets of simulated input data of the kind considered in actual emergency room situations [Kordylewski *et al.*, 1999]. The testing of the network was performed with 300 data sets (patient cases), some of which have incomplete data (in emergency-room setting there is no time for urine or other drug tests). Because of the specific requirements of the drug abuse identification problem (abuse of cannabis should never be mistakenly identified as any other drug), the training of the system consisted of two phases. In the first phase, 200 training sets were used for unsupervised training, followed by the second phase where 100 training sets were used in on-line supervised training.

The LAMSTAR network successfully recognized 100% of cannabis cases, 97% of CNS stimulants, and hallucinogens (in all incorrect identification cases both drugs were mistaken with alcohol), 98% of alcohol abuse (2% incorrectly recognized as opiates), and 96% of opiates (4% incorrectly recognized as alcohol).

(d) Detection of Epilepsy

Detection of epilepsy from EEG record via a LAMSTAR neural network is reported in (Nigam and Graupe, 2004). The EEG signals were first reprocessed to retrieve attributes of relative spike amplitude and spike rate and median filtering over time windows of 1 second. Furthermore, median filtering and polynomial enhancement was applied before entering the attribute into the neural network. Overall detection success rate for 250 EEG segments of both epileptic and non-epileptic EEG was 97.2%, while miss rate was 1.6%.

(e) Application to Assessing of Fetal Well-Being

This application [Scarpazza *et al.*, 2002] is to determine neurological and cardiologic risk to a fetus prior to delivery. It concerns situations where, in the hours before delivery, the expectant mother is connected to standard monitors of fetal heart rate and of maternal uterine activity. Also available are maternal and other related clinical records. However, unexpected events that may endanger the fetus, while recorded, can reveal themselves over several seconds in one monitor and are not conclusive unless considered in the framework of data in anther monitor and of other clinical data. Furthermore, there is no expert physician available to constantly read any such data, even from a single monitor, during the several hours prior to delivery. This causes undue emergencies and possible neurological damage or death in approximately 2% of deliveries. In [Scarpazza *et al.*, 2002] preliminary results are given where all data above are fed to a LAMSTAR neural network, in terms of 126 features, including 20 maternal history features, 9 maternal condition data at time of test (body temperature, number of contractions, dilation measurements, etc.) and 48 items from preprocessed but automatically accessed instruments data (including fetal heart rate, fetal movements, uterine activity and cross-correlations between the above).

This study on real data involved 37 cases used for training the LAMSTAR NN and 36 for actual testing. The 36 test cases involved 18 positives and 18 negatives. Only one of the positives (namely, indicating fetal distress) was missed by the NN, to yield a 94.44% sensitivity (miss-rate of 5.56%). There were 7 false alarms as is explained by the small set of training cases. However, in a matter of fetal endangerment, one obviously must bias the NN to minimize misses at the cost of higher rate of false alarms. Computation time is such that decisions can be almost real time if the NN and the preprocessors involved are directly connected to the instrumentation considered.

Several other applications to this problem were reported in the literature, using other neural networks [Scarpazza *et al.*, 2002]. Of these, results were obtained in [Rosen *et al.*, 1997] where the miss percentage (for the best of several NN's discussed in that study) was reported as 26.4% despite using 3 times as many cases for NN-training. Studies based on Back-Propagation yielded accuracy of 86.3% for 29 cases over 10.000 iterations and a miss rate of 20% [Maeda *et al.*, 1998], and 11.1% miss-rate using 631 training cases on a test set of 319 cases with 15,000 iterations [Kol *et al.* 1995].

(f) Predicting Onset of Sleep Apnea Events using Modified LAMSTAR Network

Prediction of sleep apnea and hypopnea events via the normalized LAMSTAR network of Sec. 9.8 above is described In (Waxman *et al.*, 2010), (Waxman, 2011), with sensitivity of 81% and specificity of 77% for 36 cases of severe apnea.

9.C. Predicting Price Movement in Market Microstructure via LAMSTAR[†]

Abstract

Information technology enables trading of financial products to be much more efficient and effective. In nowadays, 90% of the trading volumes of stocks come from algorithmic trading in market microstructure. In this project, we explore to apply LAMSTAR network to predict the price movement in market microstructure. More specifically, the historical price movement, the current order book statistics, as well as the previous order book statistics will be regarded as three subwords in the LAMSTAR model. The decision layer will provide the price movement predictions, which include (a) price higher than the current offer price, (b) price below the current bid price, and (c) price between the bid and offer. Experiment shows that LAMSTAR is very effective and efficient. It outperforms the three algorithms used for comparison, namely, SVM (Support Vector Machines — see: Cortes and Vapnik, 1995), BP (back propagation network — see Chap. 6), RBF (Radial Basis Function — see: Broomhead and Lowe, 1988) network in both success rate and efficiency (running time). Furthermore, we apply the LAMSTAR to analyze the most significant factors contributing to the price movement.

9.C.1. *Introduction*

With the rapid development of information technology, financial institutes are now shifting from human trading to computer trading strategies, which are also called "high frequency trading (HFT)" or "algorithmic trading" (http://en.wikipedia.org/wiki/High-frequency_trading). In essence, HFT tries to use computer algorithms to explore the profit opportunities in the financial market, and employs high speed computer to make the trade automatically. It is a field under rapid developments and attracts lots of attentions. In year 2000, HFT companies accounted for less than 10% of trading orders, while in 2011, they accounted for over 70% of daily trading volumes (http://en.wikipedia.org/wiki/High-frequency_trading). In year 2012, automated trading algorithms account for 98% of the trading volumes of government bonds, 90% of the trading volumes of equities, and 90% of the trading volumes on financial futures. There are several significant contributions of HFT to the financial market. First, it improves the market efficiency. In other words, if there is any discrepancy of the market price, the computer algorithms will find the discrepancy automatically and make the correction. Second, it increases competition, and makes trading cheaper and cheaper for the general crowd. For example, in 2000, the transaction fee of a single order is around \$20 to \$50, while the fee in 2011 drops to \$3.25 to \$10 because of the competition from HFT.

[†]Executed by Xiaoxiao Shi, Computer Science Dept., University of Illinois, Chicago, 2012.

In high frequency trading, the computer will receive the information about the market microstructure (including the current price, volume, order-book information), and predict the market movement in the next couple of seconds. Most of the time, HFT algorithms only hold the financial products for less than 5 seconds. As a result, it is very essential that the algorithm is accurate and efficient. With such an algorithm, the computer will make trades tens of thousands of a time in a day, and even a small gain on each trade can accumulate to a significant amount.

One important topic in HFT is to predict the price movement in the next several seconds or milliseconds, given the historical price movement. Note that it is a research problem different from conventional finance prediction that uses daily prices. There are some reasons:

- In HFT, the dataset is huge. For example, in a single day, the number of price movements of Apple Inc. (Ticker: AAPL) is about 150,000. If we map it to daily price movement, it corresponds to \$\$150,000/250 = 600\$\$ years of price movement (there are around 250 trading days per year).
- In traditional finance prediction, the daily price/monthly price is used as the prediction objective. Hence, it does not matter whether the prediction algorithm will have to run several hours or even a whole day to get the result. However, in HFT, the prediction has to be fast. A trade will happen in milliseconds. After that, the trading opportunity will disappear.
- In HFT, the price movement is more volatile owing to market microstructure (e.g., trading spreads, effect of large trades, etc. See Fig. 9.C.1).
- In HFT, the price movement is discrete. In NYSE, the minimal price movement is \$0.01 for stocks over \$10. This is also called the tick of the price movement. As a result, the price movement is discrete (up m ticks or down n ticks where m and n are integer). This phenomenon can be observed from Fig. 9.C.1. There is no price like \$120.74123\$ since the minimal price movement is \$0.01\$.

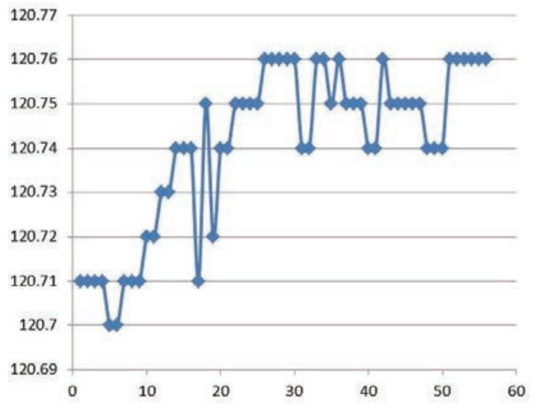


Fig. 9.C.1. Price movement of AAPL in 1.

9.C.1.1. *Objective*

In this project, our objective is to predict whether the price movement in the next 10 trades is an up-tick, a down-tick, or no significant change, given the historical price movement and the order book information. Hence, it is modeled as a classification problem with three class labels: up-tick or price higher than the current offer price, down-tick or price lower than the current bid price, and price no significant change or price between the current bid and offer.

In order to solve the problem, we employ the LAMSTAR network. Intuitively, we have three types of subwords.

- The historical trading price information;
- The current order book status (bid price, ask price, bid size, ask size, etc);
- The previous order book status (bid price, ask price, bid size, ask size, etc);

All three types of subwords will be used in making the prediction.

In the experiment section, we compare the LAMSTAR network with three commonly used classifiers: support vector machine, Back propagation network, and RBF network. The experiment shows that LAMSTAR outperforms all comparison models. Furthermore, LAMSTAR is faster than the BP and RBF network by as much as 30%, faster than SVM by as much as 55%.

We shall organize the rest of the paper as follows. We first introduce the dataset, including where it comes from, how to clean it, and the preprocessing steps. Following that, the proposed setup of the LAMSTAR network will be introduced. Then, the experiment result will be presented and discussed.

Finally, we discuss some some conclusions regarding this project.

9.C.2. ***Input Data***

In this section, we first explain the original dataset, and then introduce how we preprocess it as the input to the LAMSTAR network.

9.C.2.1. *Original Dataset*

We get the original data from Wharton Research Data Services (wrds: https://wrds-web.wharton.upenn.edu/wrds/), where UIC has a data feed subscription. More specifically, we will study the NYSE TAQ dataset in this project. The NYSE TAQ dataset contains the high frequency trading data in NYSE from 2007 to 2008. Since the dataset is very huge (over 100TB of data), we only look at the trading data and quote data (order book data) of the date 07/09/2007, and we use the highly traded stock AAPL as an example. The statistics of the selected dataset is presented in Table 9.C.1.

Table 9.C.1. Original Dataset Description

Statistics	Numbers
Number of Data in the Trade database	144,553
Number of Data in the quote database (order book)	839,789
Dimensions of the Trade database	9
Dimensions of the Quote database	9

Furthermore, there are a couple of important features in the original dataset. We explain the meanings of the features as follows.

1. Trading Price: the price traded at the moment;
2. Bid Price: the price the traders are willing to buy;
3. Ask Price: the price the traders are willing to sell; usually the ask price will be higher than the bid price
4. Spread: the difference between the ask price and the bid price, or the difference between the price the traders are willing to sell and to buy;
5. Bid Size: the amount of stocks the traders are willing to buy;
6. Ask Size: the amount of stocks the traders are willing to sell;

Although the original dataset is informative, we cannot directly use it in the LAMSTAR network for the following reasons:

- The quote database contains more data than the trade database, since the quote database reflects people's "willingness" to trade, while the trade database records the trades that "actually happen". Hence, we need an approach to combine the two dataset.
- The price is a continuous real number while in market microstructure, it is actually discontinuous. We need an approach to transform the continuous number into discrete number.
- The absolute value of bid size and ask size is not informative, one must only consider the difference between the two.

9.C.2.2. *Data Pre-Processing*

In order to perform the experiment of the LAMSTAR network, we process the dataset by matching the trading data with the order book data, and distill several features that are relevant to the task.

- First, for the quote database, we are only interested in the records before the actual trade. For example, if there is a trade happened at time 15:13:45, we are interested in the quote records at 15:13:44 since it is the quote orders that lead to the trade. We are also interested in the quote records at 15:13:45 because it is the change of the quotes immediately after the trade.

- Second, for the trade database, we are interested in the last two trading records because we assume that the market price movement follows an AR(order 2) stochastic process (Graupe, 1989). Hence, we should capture the last two trading records. The intuition of the needed information is summarized in Fig. 9.C.2.

Auto-regression information	**Trade database (time series)**	**15:13:43**	**15:13:44**	**15:13:45**
Order book information	**Quote database (time series)**		**15:13:44**	**15:13:45**

Fig. 9.C.2. Aggregated information.

- Third, we discretized the price into three categories: (1) trade at or lower than the bid price; (2) trade between the bid price and the ask price; (3) trade at or higher than the ask price. The class labels are categorized as the example in Fig. 9.C.3. In this example, any price equal to or larger than \$130.45 will be categorized as "1": trade at or higher than the ask; any price equal to or lower than \$130.35 will be categorized as "−1": trade at or lower than the bid; any price between will be labeled as "0": trade between.

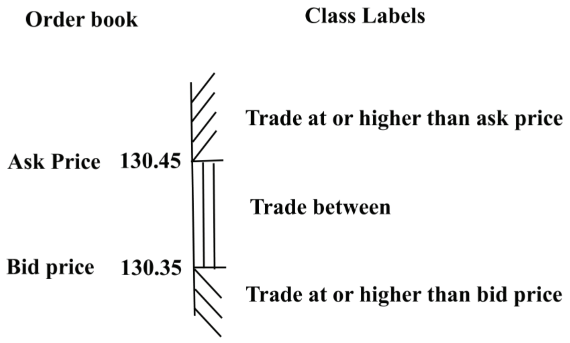


Fig. 9.C.3. Label generation.

- Fourth, we model the problem as a classification task. Hence, we summarize the average trading price of the next 10 trades, and use the label described in Fig. 9.C.3 to categorize the trades. In practice, if we predict that the next 10 trading price will be higher than the ask price, then we will take the ask price immediately, and sell at higher. Similarly, if we predict the next trading price will be lower than the current bid price, then we will hit the current bid price, and buy at a lower price.

Finally, for each of the trading data, we have the following features:

1. Current trading price (discretized as described in Fig. 9.C.2);
2. Previous 2 trading prices (discretized as described in Fig. 9.C.2);

3. Current Spread (the difference between the bid price and the ask price);
4. Current B/A Ratio (the ratio of the bid size and the ask size);
5. Current B-A size spread (the difference between the bid size and the ask size);
6. Previous Spread (the difference between the bid price and the ask price);
7. Previous B/A Ratio (the ratio of the bid size and the ask size);
8. Previous B-A size spread (the difference between the bid size and the ask size);

An example of the dataset is listed in Table 9.C.2. For the first data point, the current price is traded below the bid price (-1), and the previous trading price is higher than the ask price (1). The spread of the trade is tight (spread $= 1.05$), and there are roughly the same number of bid's and offer's (B/A ratio $= 0.8$). As a result, the buyers and sellers are very comparative, and there is no significant evidence which side is larger. In the next 10 trades, the trading price is between the current bid and ask prices (class label as category 0).

Table 9.C.2. Example of the processed data

Current price	Price @(T-1)	Price @(T-2)	Spread @(T)	B/A Ratio	B-A	Spread @(T-1)	B/A Ratio	B-A	Label
-1	1	-1	1.05	0.8	-1	0.43	1.4	2	0
0	-1	1	3.59	0.1	-9	0.33	0.33334	-2	1

9.C.3. *Setup of LAMSTAR Neural Network*

Owing to the characteristics of HFT, we need a scalable, efficient algorithm that can handle multiple sources (subword from historical data, subword from order book, subword from trading statistics, etc). Among all the neural networks, LAMSTAR seems to be a natural choice. We set up the LAMSTAR network as shown in Fig. 9.C.4. In the network, we have three layers. They represent the subwords of trading price, order book, and previous order book. As the first data point of Table 9.C.2, the vector $\langle -1, 1, -1\rangle$ is the 1st layer, the vector $\langle 1.05, 0.8, -1\rangle$ belongs to the 2nd layer, and $\langle 0.43, 1.4, 2\rangle$ is fed to the 3rd layer. The decision layer (class label) is the neuron represents 0 (trading between the bid and The 1st layer has 27 neurons. The 2nd layer has 50 neurons, where each neuron records a randomly selected sample from the training data. As in the example of Fig. 9.C.4, the 1st neuron of the 2nd layer records the vector $\langle 1.05, 0.8, -1\rangle$. The setup of the 3rd layer is similar to that of the 2nd layer.

9.C.3.1. *Training the Network*

We follow the procedure described in Chap. 9 above, the book to train the LAMSTAR network. The details are described in Fig. 9.C.5 (Algorithm I). In the experiment, we will use out-of-sample success rate to evaluate all the models since it reflects the true capability of the model in real world. More specifically, we adopt the 10-fold cross validation in evaluation. In other words, the whole dataset will be randomly split into 10 sets, and 9 sets will be used in training while the remaining will be used in testing. As a result, the input of the algorithm contains the training data as well as the testing data. Furthermore, the user can input the incremental of the link weights, with default value 0.001. The algorithm will first use the training data to update the link weights. More specifically, it will first find the winning neuron in each layer and applies the winner-takes-all principle to update the link weights. In testing, the algorithm will calculate the winning neuron in the decision layer, which serves as the class labels. Finally, the algorithm will provide the success rate of the out-of-sample testing.

9.C.4. *Results*

In the experiment, we would like to compare performance of the three approaches tested: LAMSTAR, SVM, BP and RBF regarding the present project, as follows:

- Which approach yields the best classification for the present problem?
- Which approach is the fastest for the present problem?

We also would like to answer the question:

- What are the most significant factors that can affect the price movement in the present problem?

9.C.4.1. *Comparison Methods*

In the experiment, we introduce three comparison algorithms: Support vector machine (SVM) algorithm (Cortes and Vipnik, 1995), Back propagation network (Chap. 6), and RBF (Radial Basis Function) network (Broomhead and Lowe, 1988). These are the most common choices of comparison models in the industry. Furthermore, we apply the open source software WEKA (Hall *et al.*, 2009) to perform the comparison. As mentioned in the previous paragraph, 10-fold cross validation will be used to evaluate the success rates of the models. We will also compare the running time (including training and testing) of the algorithms on a traditional desktop with 3.4G CPU and 6GB of memory.

As shown in Fig. 9.C.6, the success rate of LAMSTAR outperforms all the comparison models. In high frequency trading, a slight increment of the accuracy will mean a huge increase in the profit. Hence, the accuracy increment of LAMSTAR is very essential to improve the profit. More importantly, as the running time comparison in Fig. 9.C.7, LAMSTAR runs faster than all the comparison models. It is faster than the BP and RBF network by as much as 30%, faster than SVM by as much as 55%. As discussed in the introduction, efficiency is another key issue of high frequency trading since the trading opportunity will only exist in a very short period of time.

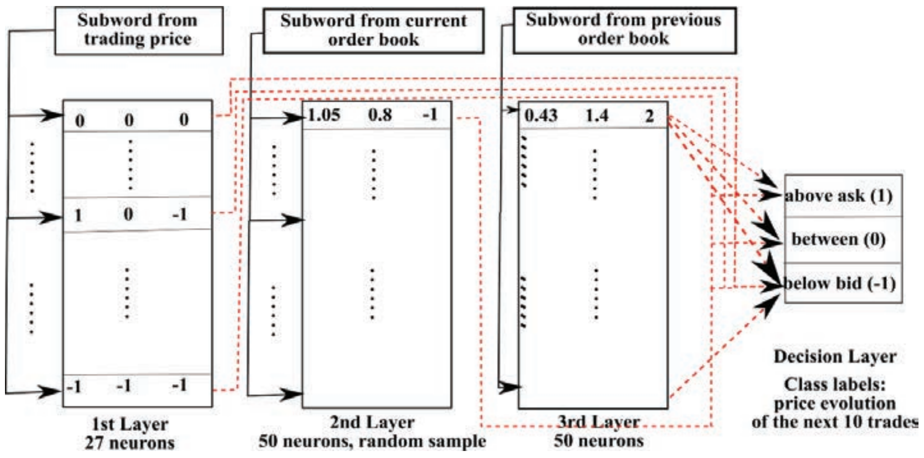

Subword from trading price
Subword from current order book
Subword from previous order book
0 0 0
1 0 -1
-1 -1 -1
1.05 0.8 -1
0.43 1.4 2
1st Layer 27 neurons
2nd Layer 50 neurons, random sample
3rd Layer 50 neurons
above ask (1)
between (0)
below bid (-1)
Decision Layer
Class labels: price evolution of the next 10 trades

Fig. 9.C.4. Setup of the LAMSTAR Network.

```
Input: Training Data, the increment $\triangle L$, the
       testing data
Output: The LAMSTAR network with trained link
        weights, success rate
1  Initialize all link weights to be 0.
2  for Each Training Data do
3      for Each Layer do
4          Calculate the winning neuron by
           minimizing the distance norm as follows
5          $d(i,j) = \|x_i - w_j\| \leq \|x_i - w_k\|, \forall k$
6          For the winning neuron, update the link
           weights by
7          $L_{i,j}^{k,m}(t+1) = L_{i,j}^{k,m}(t) + \triangle L$
8      end
9  end
10 for Each Testing Data do
11     for Each Layer do
12         Calculate the winning neuron by
           minimizing the distance norm as follows
13         $d(i,j) = \|x_i - w_j\| \leq \|x_i - w_k\|, \forall k$
14     end
15     Calculate the winning neuron in the decision
       layer by the following calculation:
16     $\sum_{kw}^{M} L_{kw}^{i,n} \geq \sum_{kw}^{M} L_{kw}^{i,j}, \forall k,j,n; i \neq n$
17     Update the success rate
18 end
19 Return the success rate and the LAMSTAR
   network
```
Algorithm 1: LAMSTAR

Fig. 9.C.5. General Outline of the LAMSTAR Algorithm.

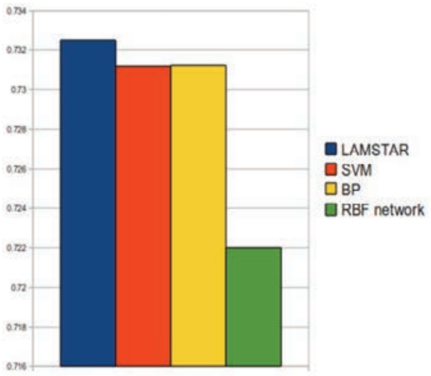


Fig. 9.C.6. Success rate (Right to Left: LAMSTAR, SVM, BP, RBF).

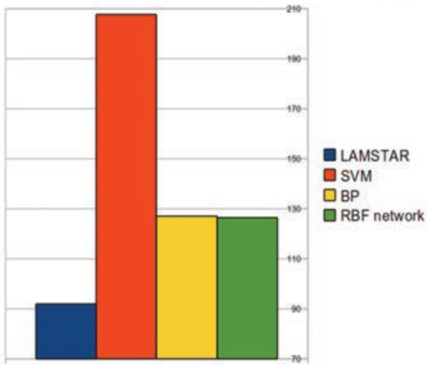


Fig. 9.C.7. Running time – comparison (Right to Left: LAMSTAR, SVM, BP, RBF).

9.C.4.2. *Most Significant Factors Analysis*

Another very important characteristics of LAMSTAR is that we can easily observe the most significant factors in a learning task. We find some interesting observations in the LAMSTAR (see: Sec. 9.7 above).

1. The most significant subword: $[i^*, s^*/dk]$: $L(i, s/dk) \geq L(j, p/dk)$ for any winning neuron $\{j\}$ in any module. It turns out that the most significant subword is the historical trading record (1st layer). It makes sense since in market microstructure; the trading price follows the autoregressive process, which is highly related to historical trading price.
2. The most significant neuron in each layer to the decision neuron -1 (trade below the bid): The most significant neuron in the 1st layer is the neuron corresponds to the vector $\langle -1, -1, -1\rangle$; that in the 2nd layer is the neuron corresponds to the vector $\langle 0.0200, 2.6667, 5.0000\rangle$; that in the 3rd layer is the neuron corresponds

to the vector $\langle 0.0200, 1.7500, 3.0000 \rangle$. They are all reasonable. For example, in the 1st layer, if the previous three trades are down ticks, it is more likely that the next trades will be down tick since it reflects a strong trend in the market; in the 2nd layer, if the buyers are more than the sellers (B/A ratio $> 1\$$), it is more likely that the next trade will be trade with the buyer's price.

3. The most significant neuron in each layer to the decision neuron 0 (trade between): The most significant neuron in the 1st layer is the neuron corresponds to the vector $\langle 0, 0, 0 \rangle$; that in the 2nd layer is the neuron corresponds to the vector $\langle 0.0900, 1.3333, 1.0000 \rangle$; that in the 3rd layer is the neuron corresponds to the vector $\langle 0.0200, 2.0000, 1.0000 \rangle$.
4. The most significant neuron in each layer to the decision neuron 1 (trade above the ask): The most significant neuron in the 1st layer is the neuron corresponds to the vector $\langle 1, 1, 1 \rangle$; that in the 2nd layer is the neuron corresponds to the vector $\langle 0.0200, 0.1429, -6.0000 \rangle$; that in the 3rd layer is the neuron corresponds to the vector $\langle 0.0200, 0.8571, -1.0000 \rangle$.

In summary, the most significant factors reflected by LAMSTAR match human intuition. For example, the historical trading price is the most important factor since it is an AR process; if there are more sellers than buyers, the next trading price is likely to be a price towards the sell side.

9.C.5. *Conclusions*

Predicting the price movement in market microstructure is a hot topic in both research and in the real world. In this project, we design a LAMSTAR to incorporate historical trading data, order book data and previous order book data to predict the price movement of the next 10 trades. We further compare LAMSTAR with 3 other classifiers SVM, BP network and RBF network. The experiment shows that LAMSTAR out-beats the comparison models in both success rate and efficiency. Furthermore, the most significant factors reflected by LAMSTAR perfectly match human intuition. It shows a great potential of LAMSTAR in the world of high frequency trading and algorithmic trading.

9.C.6. *Computer Codes – Main Outputs (LAMSTAR NN, SVM machine, BP NN, RBF NN)*

```
1. LAMSTAR
   === Run information ===
   Instances:    144552
   Attributes:   8
                V2, V4, V5, V6, V7, V8, V9, V10
   Test mode:10-fold cross-validation
   testLamstarMM

Elapsed time to build model is 88.4936008 seconds.
```

```
2. SVM
   == Run information ===
   Relation:     combine_discrete-weka.filters.unsupervised.attribute.Remove-R1,3
   Instances:    144552
   Attributes:   8
             V2, V4, V5, V6, V7, V8, V9, V10

Test mode:10-fold cross-validation

   === Classifier model (full training set) ===
   Kernel used:
    Linear Kernel: K(x,y) = <x,y>
   Classifier for classes: Bid, Between
   BinarySMO
   Machine linear: showing attribute weights, not support vectors.
    +  Number of kernel evaluations: 520653862 (39.369% cached)
   Classifier for classes: Bid, Ask
   Number of kernel evaluations: 481084673 (39.778% cached)

Time taken to build model: 207.5 seconds

3. BP network (2 layer, 4 by 2 by 1)
   === Run information ===
   Relation:     combine_discrete-weka.filters.unsupervised.attribute.Remove-R1,3
   Instances:    144552
   Attributes:   8
             V2, V4, V5, V6, V7, V8, V9, V10
Test mode:10-fold cross-validation

   === Classifier model (full training set) ===

   Sigmoid Node

   Inputs    Weights

   Threshold    -3.942077028655493

Time taken to build model: 126.94 seconds

4. RBF network

   === Run information ===
   Relation:     combine_discrete-weka.filters.unsupervised.attribute.Remove-R1,3
   Instances:    144552
   Attributes:   8
             V2, V4, V5, V6, V7, V8, V9, V10
             Test mode:10-fold cross-validation
   === Classifier model (full training set) ===
   (Logistic regression applied to K-means clusters as basis functions):
   Logistic Regression with ridge parameter of 1.0E-8

Time taken to build model: 126.37 seconds
```

9.C.7. *LAMSTAR Source Code (Matlab)*

```
function [successRate,list_predict] = lamstarNetwork_mm(trainData, trainLabel,
testData, testLabel, maxIteration)
% dimension = 12
[row,col] = size(trainData);
%numTrain = floor(9*row/10);
deltaL = [0.001, 0.001, 0.001];
deltaM = 0.00;
% forgeting rate

alpha = 1.0;
n_neurons = 50;
%% initialization from 0
L_hist = zeros(27, 3); % link weights from AR historical data to decision neurons
(-1, 0, 1)
L_orderbook = zeros(n_neurons, 3); % link weights from current order book to
decision neurons (-1, 0, 1)
L_order_pre = zeros(n_neurons, 3); % link weights from previous order book to
decision neurons (-1, 0, 1)
%% Generate the representative points for all layers
%[k,rp1]=kmeans(trainData(1:row,4:6),n_neurons);
%[k,rp2]=kmeans(trainData(1:row,7:9),n_neurons);
rp1 = zeros(n_neurons, 3);
selected = zeros(row,1);
for i=1:n_neurons
    id = floor(rand * row);
    if id==0
        id=1;
    end
    while selected(id)==1
         id = floor(rand * row);
        if id==0
            id=1;
        end
    end
    selected(id)=1;
    rp1(i,1:3) = trainData(id,4:6);
end
rp2 = zeros(n_neurons, 3);
selected = zeros(row,1);

for i=1:n_neurons
     id = floor(rand * row);
    if id==0
   id=1;
    end
    while selected(id)==1
 id = floor(rand * row);
        if id==0
        id=1;
        end
    end
    selected(id)=1;
    rp2(i,1:3) = trainData(id,7:9);
```

```
end
values=[-1,0,1];
rp0=zeros(27,3);
for a=1:3
    for b=1:3
        for c=1:3
            rp0(9*(a-1)+3*(b-1)+c,1:3) = [values(a),values(b),values(c)];
        end
    end
end
%% Start the iteration
successRate = 0;
list_predict = [];

for i=1:row
    label = trainLabel(i);
    aaa = i;
 for t=1:maxIteration
%% forgetting factor
        L_hist = alpha*L_hist;
        L_orderbook = alpha*L_orderbook;
        L_order_pre = alpha*L_order_pre;
        % Update the first layer
        diff = (ones(27,1)*trainData(i,1:3)-rp0);
        [minv, minl] = min(diag(diff*diff'));
        for j=1:3
            if abs(label-values(j))<0.5
                L_hist(minl,j) = L_hist(minl,j) + deltaL(j);
            else
                L_hist(minl,j) = L_hist(minl,j) - deltaM;
            end
        end
        % Update the second layer
        diff = (ones(n_neurons,1)*trainData(i,4:6)-rp1);
        [minv, minl] = min(diag(diff*diff'));
        label = trainLabel(i);
        for j=1:3
            if abs(label-values(j))<0.5
                L_orderbook(minl,j) = L_orderbook(minl,j) + deltaL(j);
            else
                L_orderbook(minl,j) = L_orderbook(minl,j) - deltaM;

              end
        end
        % Update the 3rd layer
        diff = (ones(n_neurons,1)*trainData(i,7:9)-rp2);
      [minv, minl] = min(diag(diff*diff'));
        label = trainLabel(i);
        for j=1:3
            if abs(label-values(j))<0.5
                L_order_pre(minl,j) = L_order_pre(minl,j) + deltaL(j);
            else
                L_order_pre(minl,j) = L_order_pre(minl,j) - deltaM;
            end
        end
    end
end
 %% Out of sample testing
[testRow, testCol] = size(testData);
list_predict = [];
for i =  1:testRow
     pred_label = zeros(1,3);
    label = testLabel(i);
        % Get the first layer
        diff = (ones(27,1) * testData(i,1:3) - rp0);
        [minv, minl] = min(diag(diff*diff'));
        pred_label = pred_label + L_hist(minl,1:3);
    % Get the second layer
        diff = (ones(n_neurons,1)*testData(i,4:6)-rp1);

  [minv, minl] = min(diag(diff*diff'));
        pred_label = pred_label + L_orderbook(minl,1:3);
    % Get the 3rd layer
        diff = (ones(n_neurons,1)*testData(i,7:9)-rp2);
        [minv, minl] = min(diag(diff*diff'));
        pred_label = pred_label + L_order_pre(minl,1:3);
      [maxv, maxLabel] = max(pred_label);
    pred_label = 0;
    if maxLabel == 1
        pred_label = -1;
    end
    if maxLabel == 3
        pred_label = 1;
    end
    if abs(label-pred_label)<0.5
        successRate = successRate+1;
end
    list_predict = [list_predict; [pred_label, label]];
end
successRate = successRate / (testRow);
```

9.D. Constellation Recognition[‡]

9.D.1. *Introduction*

The aim of this project is to develop a system capable of find the presence of a constellation in an image and recognize it. Some system of this kind has already been developed but their approach difference from this in many ways. The most important difference between this system and others is the fact that usually this kind of systems aim to recognize a single star by the position of some of the closer stars. These systems are used on satellites in order to precisely determine their orientation.

Systems similar to the one presented here are used on space modules like satellites in order to determine their orientation, this is a very difficult and important

[‡]Executed by Giovanni Paolo Gibilisco, Computer Science Dept., University of Illinois, Chicago, 2010.

task because the correct orientation of the satellite with respect to earth affects the navigation and the efficiency of the communication system. Here is a brief example of how these systems work. The satellite has a CCD sensor which takes images of the stars. These images are then processed in order to find the name of the star that has been photographed. The position of the recognized star in the image is used to calculate the orientation of the satellite according to the position of the star read from a database. In order to correctly identify the brightest star in a field of view of the CCD, the system calculates relative measures between the positions of some stars very close to the one to be identified.

9.D.1.1. *Problem definition*

The scope of this system is to recognize constellations. The main differences from the system described above and the one implemented in this project are that the images used for the project are taken from earth, not from satellites, and that the system is focused on the identification of an entire constellation, not only on a single star. The fact that images are taken from a camera on earth and not from a very expensive and precise sensor positioned on a satellite is one of the main challenges of this project.

9.D.2. *Approach*

The approach presented here for the processing of images is divided into two main steps:

(i) Preprocessing of the original image and feature extraction. (ii) Test of the preprocessed image and the features on a neural network based system. Preprocessing is then divided into two main parts: Image preprocessing and feature extractions. Image preprocessing consists of manipulating the original image in order r to reduce the noise that is due to many factors like the low quality of the image, the variation of luminosity of some part of the image due to the effect of earth's atmosphere. The second part is extraction of features from the preprocessed image in order to provide the neural network significant data on the extracted features, such as the average distance between the stars in the image. Figure 9.D.1 shows the basic structure of the system.

9.D.2.1. *Image Preprocessing*

This paragraph deals with the preprocessing of the image in order to filter noise due to many factors that occurred when the picture was taken. The result of the preprocessing will be an image similar to the original one but that can be better recognized by the neural network based system.

Noise factors: Preprocessing of star of images taken from earth is very problematic. There are a lot of factors that influence the quality of an image; some of those factors are discussed below.

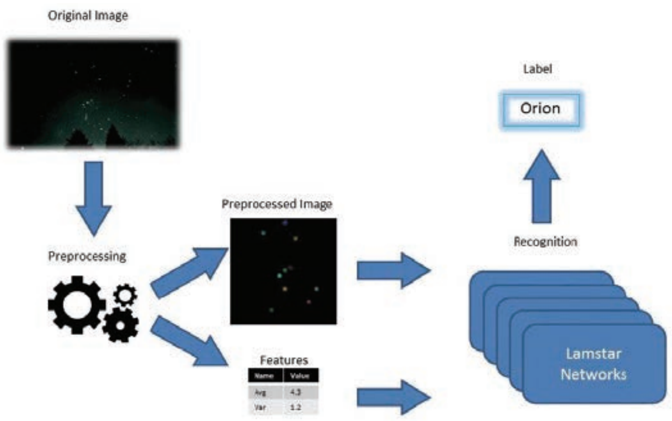


Fig. 9.D.1. System's Basic Structure.

Exposure time is an important factor to take into account while processing star images. The main difference between normal photographs and star images is that they require a longer exposure time. In order to take an image of the sky in which many stars are visible the CCD sensor of the camera need to stay opened for a long period of time, during this time every movement of the camera can corrupt the final image. During this period of time the camera need to point exactly the portion of sky one is trying to capture, if we are using a normal lens the rotation of earth that occurs during the exposition time is visible on the final image and the stars will not look like bright points but as curved lines. This effect is even more evident if we are taking a photo using a telescope because the field of view of the CCD will be smaller and the movement appears faster. To compensate this effect usually one has to use a motorized trestle that make the camera move accordingly to the portion of the sky pointed by the camera objective.

Curved Images as mentioned above can hardly be used for constellation recognition because it's impossible to reconstruct a steady image of the sky. Hence, we limit our discussion to earth images (Fig. 9.D.2).

Earth atmosphere and **other sources of light** are the most important factors to take into account during the processing of these images. The presence of earth atmosphere reduces the quantity of light coming from the stars that reach the objective of the camera and expand any other source of light coming from earth. So the presence of a city near the place where the picture is taken has a great impact on the luminosity of the image. In the next section is shown how a simple algorithm can partially overcome this problem.

Even if the image has been captured with particular attention to the condition of the environment another problem that is crucial to constellation recognition can occur, this problem is the presence of a large number of minor stars. Usually in a picture of a portion of the sky some stars with low brightness appear. Due to effect of the atmosphere and to the structure of the human eye this stars are usually not

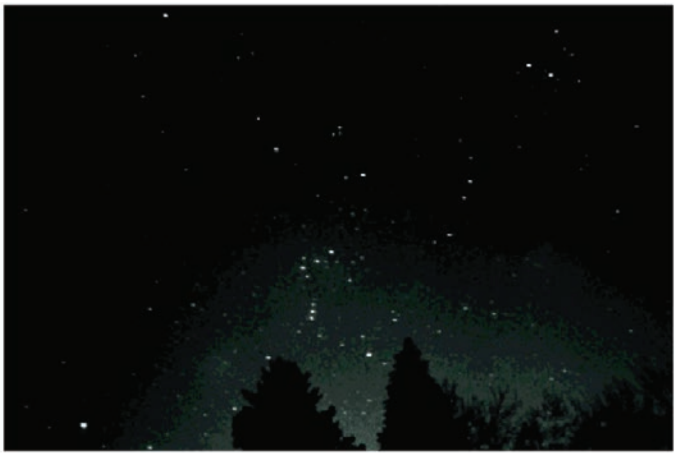


Fig. 9.D.2. Image of Sky Section Taken from Earth.

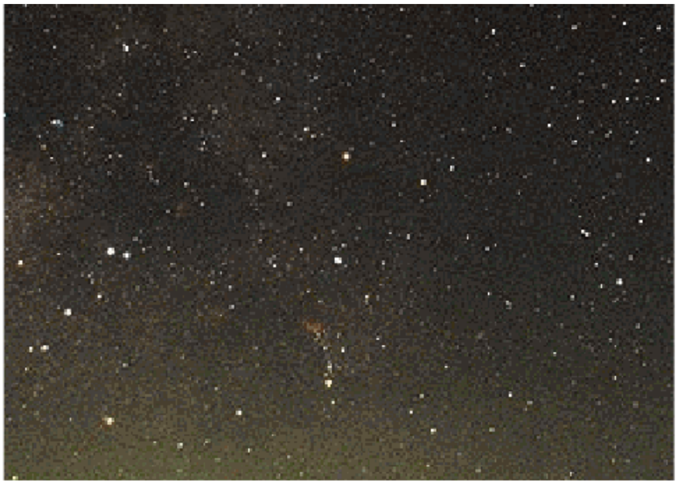


Fig. 9.D.3. Unfiltered Sky Image.

visible, but the long exposition time used to capture images make this star appear more brilliant than they seem to humans. Constellations are composed only of stars that are easily visible from earth and any other star that appear in the image due to this effect can be considered as noise. An example of this can be seen in Fig. 9.D.3.

Images that used for this project come from different websites and are affected by all of these problems.

Step 1 – Sparse luminosity reduction: The first step in the processing of these images is concerned with reduction of the sparse luminosity due to earth atmosphere. Figure 9.D.4 is an image that is heavily affected by this problem, while the outcome of this preprocessing step is shown in Fig. 9.D.5. This preprocessing stage is as follows: Initially, the image is scanned and the average luminosity is calculated. The average luminosity of the image is a good estimator of this kind of problems. If the image contains a very dark sky with a few bright stars, that should be the ideal case, then the average is relatively low because the pixels that are affected by the luminance of the star are very few. However, if the sky in the image is very bright then the average will be high. The average alone is not a perfect indicator of this problem because some pictures may have a very dark sky but a lot of small stars that are very bright. In order to deal with this kind of images. the preprocessing algorithm computes also the root mean square (RMS) of the luminosity on all the pixels. The image is then filtered in order to reduce the luminosity of all the pixels that are below a threshold. If the luminosity of a pixel is below that threshold then its color is changed to black. The outcome of this preprocessing step is shown in Fig. 9.D.4.

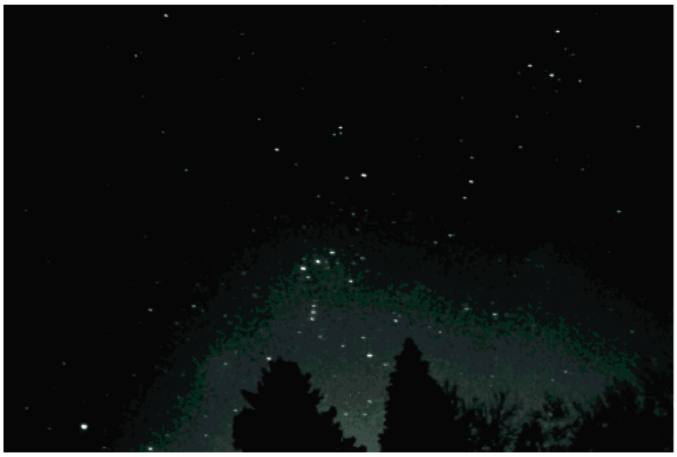


Fig. 9.D.4. Before Averaging-Out Background of Sky.

The outcome of this preprocessing step is a photo in which the average luminosity is lower than the original one and the variance is higher. This approach is very useful when dealing with photos that have been taken near a city or another major source of light. There are two main problems with this approach. The first is that the setting of the threshold. The second, and more important problem is that this filter is able to reduce the sparse luminosity but does not change the luminosity of the stars in the picture. As we shall see later, this is a major problem when extracting relative measures from the picture.

Step 2 – Small star filtering: Not all the photos considered in this project are affected by this problem, some other have a very dark sky but a lot of small and very bright stars. These photos are usually taken from satellites or from very dark places on the earth using a long exposure time. The main problem with these photos is the fact that they have too many stars. Constellations has been named by man by

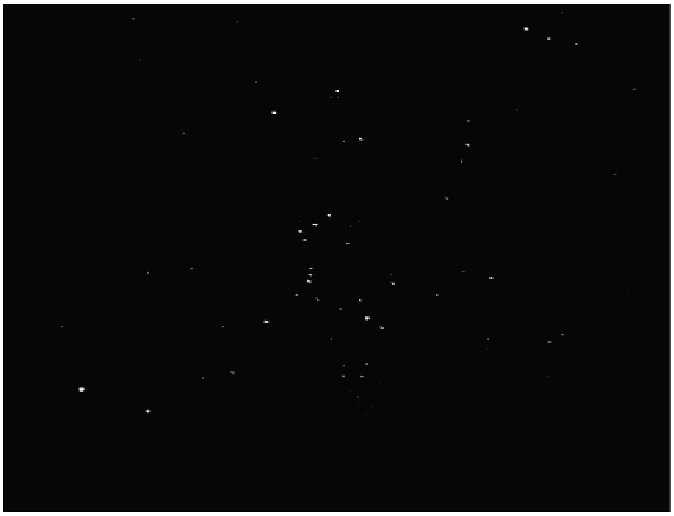


Fig. 9.D.5. After Averaging of Background Sky.

looking at some very bright stars in a portion of the sky, many of the stars present in Fig. 9.D.6 are not visible without some optical tool like a binocular or a telescope. These stars introduce a lot of noise in the image. The example in Fig. 9.D.7 shows a photo of the Ursa Major constellation surrounded by many other stars. In this picture the stars that are part of the constellation are the just seven, while the total number of stars present in the picture is higher than a hundred. In order to filter out the less important stars, a second filter is created. This filter is based on the average and the variance of the luminosity computed on the entire image. The re-computation of these two measures is necessary because the previous filter altered some pixels in order to make the image darker. The filter scans each pixel of the image and calculates the average and variance luminance of a small area, currently a square of length 5 pixels. Consequently, if these values are above a threshold then the pixel left untouched otherwise it's changed to be black. The effect of this filter (see Fig. 9.D.7) is visible on all the stars composed by a few number of pixels. Stars that appear bigger, though more important in the identification phase, are poorly affected by this filter since it eliminates only a few pixels on the outer part of the star. This filter relies on the fact that a small star loosely affects the pixels surrounding it. In some images taken from earth, the atmosphere spans the star's luminosity also in adjacent pixels of the image that the filter presented with in the previous paragraph helps to reduce this sparse luminosity and makes this second preprocessing step more efficient. As we can see in Fig. 9.D.7, the overall quality of the image with respect to the recognition process is quite high, the main problems now are the orientation of the picture, the size of the image and the relative high number of star that are still in the picture.

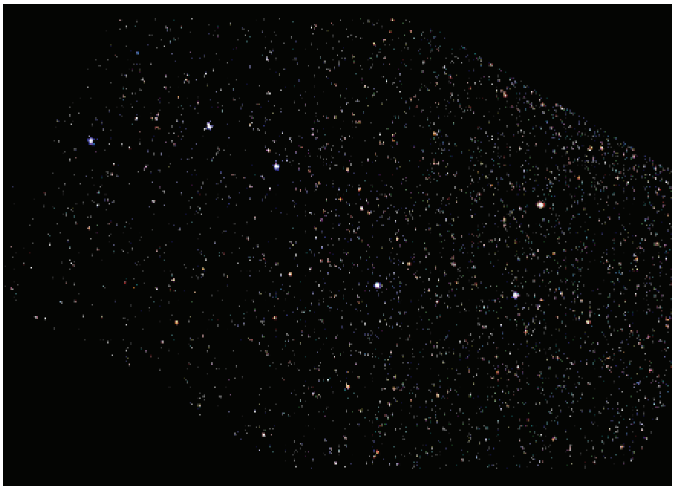


Fig. 9.D.6. Before Small-Star Filtering.

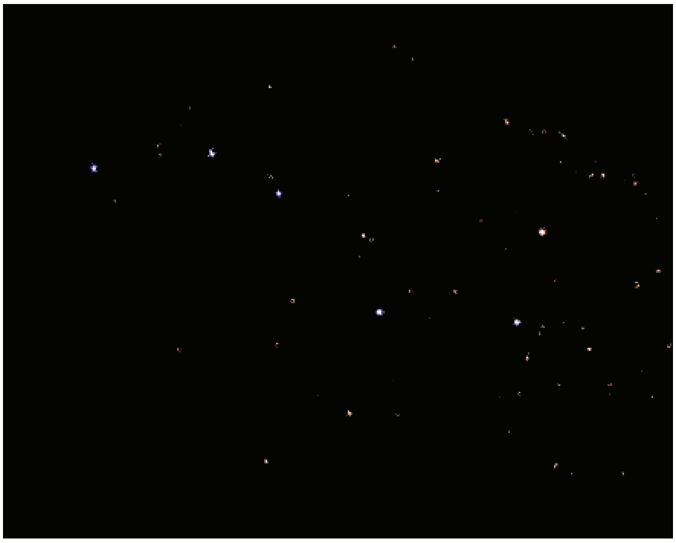


Fig. 9.D.7. After Small-Star Filtering.

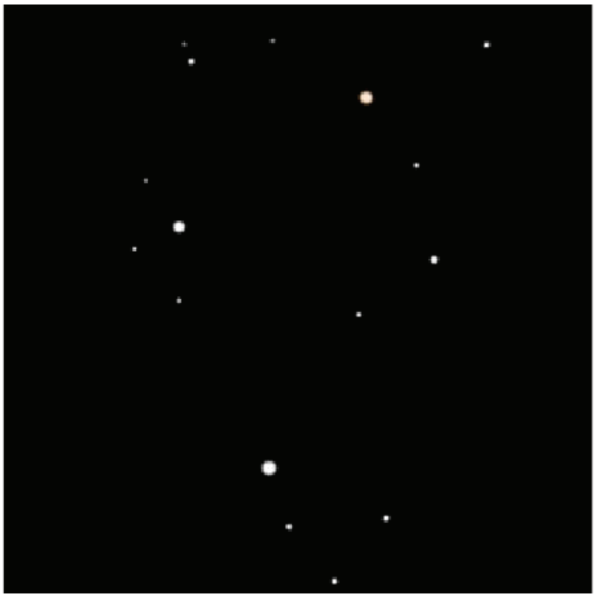
Step 3 – Rotation and reduction: A very important problem in the identification of constellations is the fact that the photo we are using has been taken from different part of hearth at different time, so a constellation can appear rotated or shifted. The usual way to rotate this kind of picture is to find the two brightest stars and align them. This method is very efficient is used from images taken from satellites with high precision sensors that measures directly the luminance of the stars in their field of view. Unfortunately this is not feasible for images taken from earth with a normal camera. The presence of a source of light like a distant city or the moon usually modify the luminance of an area of the image in such a way that even small stars that are close to this area seems brighter than other bigger stars at the opposite part of the image. This problem is now solved by manually rotating the image. This approach is feasible for a small training set like the one used here while it's not applicable to lager sets.

In order to use these pictures as an input to a neural network one need to scale them so that they have the same length. This is done in two steps. In the first the image is enlarged in order to make it squared by adding horizontal or vertical black bars. Then this squared picture is reduced in order to fit in a 40×40 matrix. The reduction of the image is done by dividing the image into 40 rows and 40 columns and by calculating the average of the RGB value of the pixels in the area in order to calculate the color for the final reduced image. Using this method allows us to maintain the different color of the stars and their luminance. In order to filter out some of the stars that the previous filter was not able to filter another threshold is applied to the newly generated pixels. If their luminosity is lower than a value that depends on the average and variance of the luminosity of the entire image, than the pixel is set to be black. This filter may seem equal to the one described in step 2, but it is not. The main difference between the two filters is that the previous one works on a smaller area and affect also the brightest stars while these work on all the pixels of the original image that are going to be represented by the same pixels in the reduced image and doesn't affect the brightest stars. This filter is very effective if applied after filters described in steps 1 and 2. The output of this preprocessing step is shown in Fig. 9.D.8b.

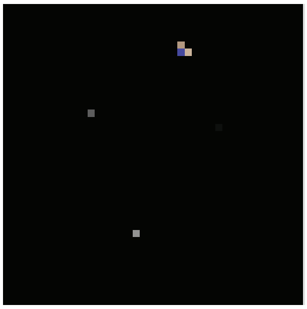
Step 4 – Patter creation and feature extraction: In order to reduce the dependency of the image recognition from the orientation of the initial image and from the false luminosity due by earth atmosphere two other steps are performed. The first is to extract two features which are the average and variance of the distance of the stars in the image; these features are saved in separate sub words that will be used in the recognition phase. The second step is to create patterns by connecting the closest stars. In order to do so the algorithm implemented connects each star to its closest one if it has not already been connected to other stars. This is done in order to provide the network a more informative picture. This preprocessing step resolves the problem of a very low number of pixels that are different from zero in the image generated by step 3. The outcome of this process iscan shown in Fig. 9.D.9.

9.D.3. *Recognition*

In order to recognize the constellation, different network have been tried in this study. The final choice has been the Lamstar network. This network has been



(a) Scaling: Stage 1



(b) Scaling: Stage 2

Fig. 9.D.8. PREPROCESSING — Scaling Stages.

chosen because of its ability to use information derived from the preprocessing phases along with the image, in order to recognize the picture. This paragraph described the set of images chosen to form the training and testing set and the structure of the network.

9.D.3.1. *Training and testing set*

In order to develop the system, 5 constellations have been chosen. For each constellation a different number of pictures, from 4 to 7, has been taken into consid-

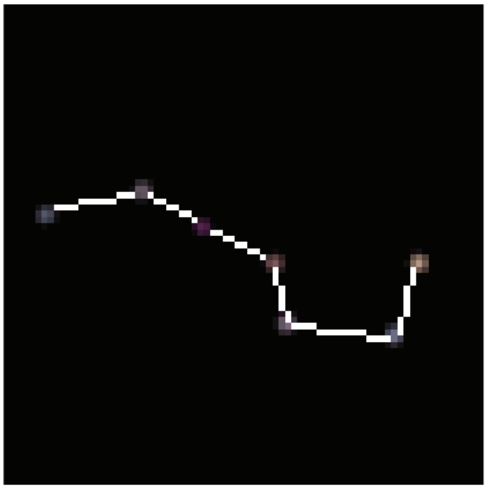


Fig. 9.D.9. Preprocessed Image (with inserted connection between selected pixels).

eration. The previously described preprocessing has been applied to all the images and the result of the preprocessing has been saved. Most of the time used to create the network is spent during preprocessing of the images so, to avoid a long training time, the result of the preprocessing phase has been saved into some image files. A complete dataset of all the images considered has been created. Subsequently, the datasets for training and testing have been automatically generated from this complete dataset by randomly choosing some pictures and by maintaining constant the number of pictures per constellation in both the training and testing sets. In order to enrich the testing, both the training and testing sets have been expanded by adding a few bits of noise. The addition of a few bits of noise in this stage has been done in order to simulate an error in the preprocessing step. The original images are already very noisy. The analysis of robustness of the system with respect to noise is very difficult because it would require many clear original images. Also, the automatically generated noise would be different with from the one present in real images.

9.D.4. *System Architecture*

The system is divided into 3 main parts: Preprocessing, NN array and output determination.

The preprocessing steps described in the previous section generate the input for the network array. The array is composed by one Lamstar network per each constellation that the system will recognize. This has been done in order to train more specifically the single networks and to gain accuracy. The training phase is the same for each network. However, the output neuron of each network is rewarded only for the constellation that the particular network will recognize.

Each LAMSTAR network is dynamically built during the training phase. Each network has 40*40 + 2 sub words.

9.D.4.1. *Lamstar Architecture*

The LAMSTAR neural network is well suited for statistical analysis and classification. The basic structure of a LAMASTAR network is represented in Fig. 9.D.10. It consists of an SOM module for each sub word that is presented at the input to the network and of an output layer and of related LINK-WEIGHTS between "winning" neurons from the various many layers. The particular structure of the network and its link-weights make it easy to add features for evaluating the images. In order to add a new feature in the evaluation we add a new SOM module corresponding to the new input subword and establish its link weights. In this project, the LAMSTAR serves to recognize images using the raw image (as a matrix of continuous values from 0 to 1, or as a binary matrix), and features extracted from the various preprocessing stages.

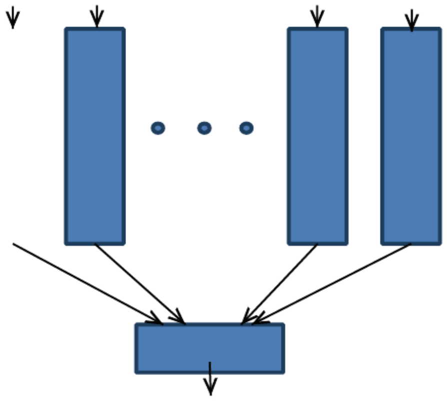


Fig. 9.D.10. LAMSTAR's Basic Structure: input layers at top; output layer(s) at bottom.

The main task of the Lamstar network is to recognize the most important features for the recognition of the pattern in the image. This ability allows one to present as input all the features that the preprocessing steps can extract, while the network itself serves to decide which features are the most significant and discard the others.

This capability is due to the fact that the link between the output layer and the neurons in all the SOM modules are rewarded when the expected output of the network is positive and punished on the opposite case. The SOM modules act as memories that recognize patterns inside their specific sub word. The first 40*40 sub words have been derived from the image by sampling all the columns and rows. The last two sub words are each composed by a number which represents the result of the computation of the specific feature obtained after preprocessing.

At the beginning of the training each SOM module is empty, the training of the SOM modules proceed as follows:

- The new sub word is presented as in put to the SOM and all the neurons in the SOM fire.
- If any neuron fires, then the SOM module is left unchanged. If no neuron fires, then a new one is added, its weights are set randomly within -0.5 and 0.5 and are then iteratively adjusted until the output of the neuron converges to one, within a given threshold.

Dynamically setting neurons in the SOM modules reduces considerably the number of neurons of the final network and doesn't affect much the average accuracy. The average number of neurons created during the dynamic training is near one third of the total number of neurons used for the static training. This reduction of the number of neurons in the network also improves the training and testing time.

The training of the output layer is performed in a different way since the number of output neurons in this layer is fixed. Since the system is composed by one network for each constellation the output layer of these networks contain just one neuron. This neuron is connected with all the other neurons in the network via link weights. The training of these link weights follows a reward/punishment principle.

Further improvement in the recognition speed could be achieved by removing all the neurons that are connected to the output neuron via a link weight with weight equal to 0, or below a threshold value.

9.D.4.2. *LAMSTAR Array*

The system consists of five Lamstar networks, one per each constellation. During the test phase, each network is presented with the same inputs which are extracted from the image by the preprocessing steps described above. If the output vector contains only one value equal to 1, the image has been recognized and the label of the network that produced the positive output is assigned to the tested image. If the array contains multiple ones it means that more than one network recognized the image. The decomposition of the system in multiple networks allows us to reach a few conclusions on the kind of error that the network produces and possibly correct it. If no network recognized the picture it means that most likely this image contains no constellation or contains a constellation that wasn't in the training set. This also allows the system to recognize images without constellation even if this category was not used during the training of the networks. In the current implementation if multiple networks classified the image then no label is assigned to the image but two simple strategies could be used to improve performances. A simple strategy could be to assign randomly a label from the network that classified them; another more fine strategy could be to retrieve the continuous values produced by the output neurons of the networks and make the vector act according the winner take all principle. In order to do that it should be necessary to remove the threshold that is currently present at the output of each network.

9.D.5. *Results*

The dataset of 29 images of 5 constellations has been preprocessed and expanded in order to obtain a dataset in which each constellation is represented by the same number of images. This dataset has been then divided into training and test set. The network has been finally trained on some images and tested against unseen examples. The main challenge of this dataset is the low number of images per constellation (from 4 to 7) and the presence of different kind of noises in the original images as described before.

The repeated testing of the system has yielded an average accuracy of 70% for unseen images. Noting the very limited data set considered we used, as stated above, this success rate is not at all low.

9.D.6. *Conclusions*

The outcome of this case study indicates the validity of a LAMSTAR neural network-based approach in the recognition of star patterns. In order to further improve this application, one needs to improve the performance of this system, mostly by expanding the training set and adding other features extracted from the image. The main problem with the extraction of the image is the fact that the sparse luminosity present in many pictures alters the luminosity of stars in different areas of the picture.

9.D.7. *Source Code (Java)*

1. Constellations

```
package networks;
public enum Constellation {
URSA_MAJOR, ORION, SCORPIUS, LEO, CRUX;//, CORONA_BOREALIS;
public static int[] bitValue(Constellation c){
int[] value = new int[(int) Math.ceil(Math.log(Constellation.values().length)/
        Math.log(2))];
char[] charValue = Integer.toBinaryString(c.ordinal()).toCharArray();

                         //get the value and add zeros if needed.
for(int i=0; i<value.length; i++)
if(value.length - charValue.length > i)
value[i] = 0;
else
value[i] = Integer.parseInt(""+charValue[i-(value.length-charValue.length)]);
//debugging..
/*System.out.print("Constellention enum->bit converison. Value: "+c.ordinal()+"
        Bit: ");
for(int i=0; i<value.length;i++)
```

```
System.out.print(value[i]);
System.out.println();
*/
return value;
}
public static Constellation nameFromBitValue(int[] value){
int orderValue=0;
//value = sum over all i of i*2^i
for(int i=0; i<value.length;i++){
orderValue += Math.pow(2, i)*value[i];
}
//retrive the name
Constellation c = Constellation.values()[orderValue];
//debugging..
System.out.print("Constellention bit->enum converison. Bit: ");
for(int i=0; i<value.length;i++)
System.out.print(value[i]);
System.out.print(" value: "+c);
System.out.println();
return c;
```

2. LAMSTAR - JAVA

```
package networks.lamstar;

//Just an interface to present inputs to the real network.
public class InputNeuron extends Neuron {
                double output;
public InputNeuron() {
super();
}
void init(double val){
output = val;
}
double getOutput(){
return output;
}
```

3. Layers

```
package networks.lamstar;
import java.util.ArrayList;
import networks.Constellation;
public class Layer {
ArrayList<Neuron> neurons = new ArrayList<Neuron>();
//add neuron until there are enough to recognize all categories
public Layer(int neuronNumber) {
for(int i=0 ;i < neuronNumber;i++ ){
neurons.add(new Neuron());
}
}
//connect all neurons to all som modules with random low weights [-0.5, 0.5]
public void connect(Neuron input){
for(Neuron out_n: neurons)
out_n.addInput(input);
}
```

```
public void fire(){

               for(Neuron n: neurons){
n.evolve();
}
}
public boolean train(int[] expected){
int error = 0;
for(int i=0;i< expected.length; i++){
//if the expected output of the single neuron is wrong punish it
if(expected[i] == 0){
neurons.get(i).punish();
//otherwise reward it
}else{
neurons.get(i).reward();
}
if(expected[i] != neurons.get(i).getOutput())
error++;
}
if(error > 0){
// printWeights();
return true;
}
return false;
}
public boolean getError(int[] expected){
for(int i=0;i< expected.length; i++)
//if the expected output of the single neuron is wrong punish it
if(expected[i] != neurons.get(i).getOutput()){
//System.out.println("expected: "+expected[0]+" "+expected[1]+" found:
                "+neurons.get(0).getOutput()+" "+neurons.get(1).getOutput());
return true;
}


                return false;
}
public void printWeights() {
for(Neuron n: neurons){
System.out.println("neuron: "+neurons.indexOf(n));
for(Neuron in: n.getInputs())
System.out.println(n.getWeight(in));
System.out.println();
}
}
public void test() {
this.fire();
for(Neuron n: neurons){
for(Neuron in: n.getInputs()){
if(in.getOutput() == 1 && n.getOutput() == 1){
n.updateWeight(in, n.getWeight(in)+Network.REWARD);
}else{
n.updateWeight(in, n.getWeight(in)-Network.PUNISHMENT);
}
}
}
}
public boolean train(Constellation image, Constellation focus) {
int error = 0;
if(image == focus)
neurons.get(0).reward();
else{
neurons.get(0).punish();
error++;
}
if(error > 0){

// printWeights();
return true;
}
return false;
}
}
```

4. Networks

```
package networks.lamstar;
import java.util.ArrayList;
import java.util.Collections;
import networks.Constellation;
import networks.Image;
public class Network {
public static final double INITIAL_SOM_LEARNING_RATE = 0.7; //0.7
public static final double SOM_CONVERGENCE_TH = 0.01; //0.05
public static final double FEATURE_TH = 0.01;
public static double INITIAL_SOM_THRESHOLD = 0.05; //0.05
public static double REWARD = 0.1;
public static double PUNISHMENT = 0.1;
public static double NOISE = 2;
ArrayList<SOMmodule> somModules = new ArrayList<SOMmodule>();
Layer outputLayer;
boolean singleOut = false;
Constellation focus = null;
public Network(int subwordNumber, boolean singleOut) {
this.singleOut = singleOut;
if(singleOut)
outputLayer = new Layer(1);
else
outputLayer = new Layer(Constellation.bitValue(Constellation.values()[0]).length);

//create one SOM module for each subword
for(int i =0 ;i < subwordNumber; i++)
somModules.add(new SOMmodule());
//add the output module
}
public Constellation getFocus(){
return focus;
}
//trains the network using the whole set of example
public void train(ArrayList<Image> examples,Constellation focus){
this.focus = focus;
```

```
examples = expandTraining(examples);
System.out.println("training set size after expansion: "+examples.size());
for(Image i: examples){
//System.out.println("image: "+examples.indexOf(i));
//i.binary();
//System.out.println(i.getName());
//i.printMatrix();
ArrayList<double[]> subwords = i.getSubwords();
//store the subword in the SOM modules
for(SOMmodule layer: somModules){
Neuron newNeuron = layer.storeSubword(subwords.get(somModules.indexOf(layer)));
if(newNeuron != null){
outputLayer.connect(newNeuron);
}
}
//train the output layer
outputLayer.fire();

if(singleOut)

                                                        outputLayer.train(i.getName(), focus);
else
outputLayer.train(i.getExpectedResult());
}
int neurons = 0;
for(SOMmodule som: somModules){
neurons += som.neurons.size();
}
System.out.println("neurons in som modules: "+neurons);
System.out.println("end train");
}
public double test(ArrayList<Image> test_set){
double error = 0;
for(Image i: test_set)
if(classify(i))
error++;
// System.out.println("errors: "+ error+" on "+test_set.size());
error = (error*100)/test_set.size();
return error;
}
//fires the network (return true if error, false if not error)
public boolean classify(Image figure){
ArrayList<double[]> subwords = figure.getSubwords();
//recognize subwords
for(SOMmodule layer: somModules)
layer.fire(subwords.get(somModules.indexOf(layer)));
//classify
outputLayer.fire();
if(singleOut)
if(outputLayer.neurons.get(0).getOutput() == 1)


                                                                        return true;
else
return false;
```

```
boolean error =  outputLayer.getError(figure.getExpectedResult());
/*if(error){
System.out.println("wrong on: ");
figure.printMatrix();
}*/
return error;
}
private ArrayList<Image> expandTraining(ArrayList<Image> examples) {
/*for(Image i: examples){
i.binary();
i.emphasize();
System.out.println("image:");
i.printMatrix();
}*/
ArrayList<Image> training = (ArrayList<Image>) examples.clone();
int noiseBits = (int) NOISE*examples.get(0).getFigure().length*examples.get(0).
        getFigure().length/100;
int x,y;
// System.out.println(training.size());
for(Image image:examples){
for(int j=0;j<5;j++){
double[][] tmp = new double[image.getFigure().length][image.getFigure().length];
for(int k=0;k<tmp.length;k++)
for (int l = 0; l < tmp.length; l++)
tmp[k][l] = image.getFigure()[k][l];
for(int i = 0;i<noiseBits;i++){
x = (int) (Math.random()*(image.getFigure().length-1));
y = (int) (Math.random()*(image.getFigure().length-1));
if(tmp[x][y] == 1)


else
tmp[x][y] = 1;
}
// System.out.println("after");
// image.printMatrix();
training.add(new Image(tmp, image.getName()));
}
}
Collections.shuffle(training);
return training;
// System.out.println(training.size());
}
```

5. Neuron

```
package networks.lamstar;
import java.util.ArrayList;
import java.util.HashMap;
import javax.swing.text.MutableAttributeSet;
public class Neuron {
ArrayList<Neuron> inputs;
HashMap<Neuron, Double> weights;
double output;
public Neuron() {
inputs = new ArrayList<Neuron>();
weights = new HashMap<Neuron, Double>();
output = 0;
}
void addInput(double weight, Neuron input){
inputs.add(input);
weights.put(input, weight);
}

ArrayList<Neuron> getInputs(){
return inputs;
}
public double getWeight(Neuron n){
return weights.get(n);
}
//fires the neuron
void evolve(){
output = 0;
//internal sum
for(Neuron n:inputs)
output += n.getOutput()*weights.get(n);
//function
if(output > 0)
output = 1;
else
output = 0;
}
double getOutput(){
return output;
}
public void updateWeight(Neuron n, double weight) {
weights.put(n, weight);
}
public void addInput(Neuron inN) {
this.addInput(Math.random() -0.5,inN);
}
public void punish() {
for(Neuron in: inputs)



                      if(in.getOutput() == 1)
updateWeight(in, weights.get(in)-Network.PUNISHMENT);
}
public void reward() {
for(Neuron in: inputs)
if(in.getOutput() == 1)
updateWeight(in, weights.get(in)+Network.REWARD);
}
```

6. SOM Module

```
package networks.lamstar;
import java.util.ArrayList;
public class SOMmodule {
ArrayList<SOMneuron> neurons = new ArrayList<SOMneuron>();
ArrayList<InputNeuron> inputNeurons = new ArrayList<InputNeuron>();
```

```
//trains the SOM module for the given subword adjusting weights or creating new
    neurons if needed.
public Neuron storeSubword(double[] subword)
//if the subword is all zero return
double sum=0;
for(int i=0; i< subword.length; i++)
sum += subword[i];
if (sum == 0)
return null;
//if the layer is empty add the first neuron
if(neurons.isEmpty()){
this.fillInputNeurons(subword);
return this.addNeuron(subword);
}
//else fire the layer and look for an output equal to 1
this.fire(subword);


                               //if it exists
for(SOMneuron n: neurons)
if(n.output == 1)
return null;
//else create a new neuron
return this.addNeuron(subword);
}
//creates the input neuron according to the length of the subword
private void fillInputNeurons(double[] subword) {
for(int i=0; i< subword.length;i++)
inputNeurons.add(new InputNeuron());
}
//fires all the neurons in the module and check that either one or no neuron has
    output 1
public void fire(double[] subword) {
//init input neurons with the subword
for(InputNeuron in: inputNeurons)
in.init(subword[inputNeurons.indexOf(in)]);
//evolve each neuron
int ones = 0;
for(Neuron n: neurons){
n.evolve();
ones += n.getOutput();
}
//notify if more than 1 neuron fires
/*if(ones > 1)
System.err.println("More than one neuron fired in layer "+this);
*/
}
//add a neuron to the SOM layer
private Neuron addNeuron(double[] subword) {
//init input neurons with the subword

for(InputNeuron in: inputNeurons)
in.init(subword[inputNeurons.indexOf(in)]);
SOMneuron tmp = new SOMneuron();
//connect all the input neurons
```

```
for(InputNeuron in: inputNeurons)
tmp.addInput(in);
//set the weights
tmp.updateWeight(subword);
//add to the layer
neurons.add(tmp);
return tmp;
}
public ArrayList<SOMneuron> getNeurons() {
return neurons;
}
```

7. SOM Neuron

```
package networks.lamstar;
public class SOMneuron extends Neuron{
double threshold = Network.INITIAL_SOM_THRESHOLD;
double learningRate = Network.INITIAL_SOM_LEARNING_RATE;
//fires the neuron
void evolve(){
if(inputs.size() == 1){
if(Math.abs(inputs.get(0).getOutput() - weights.get(inputs.get(0))) <=
            Network.FEATURE_TH)
output = 1;
else
output = 0;
return;
}
output = 0;

                                //internal sum
for(Neuron n:inputs)
output += n.getOutput()*weights.get(n);

                               //non linear function
if(Math.abs(1 - output) < threshold)
output = 1;
else
output = 0;
}
public void updateWeight(double[] subword) {
if(subword.length == 1){
updateWeight(inputs.get(0), subword[0]);
return;
}
do{
for(Neuron n: inputs){
//the new weight is w(n+1) = w(n) + alpha*[x-w(n)]
updateWeight(n, weights.get(n) + learningRate*(subword[inputs.indexOf(n)] -
                weights.get(n)));
//normalize new weights
this.normalizeWeights();
}
//diminuish the learining rate
//learningRate /= 2;
if(learningRate < 0.1){
```

```
learningRate = Network.INITIAL_SOM_LEARNING_RATE;
}
this.evolve();
//System.out.println("output = "+this.getOutput());
}while(Math.abs(1-this.getOutput()) > Network.SOM_CONVERGENCE_TH);
}

       public void normalizeWeights(){
//calculate what to divide by
double div = 0;
for(Neuron n: inputs)

                                div += weights.get(n)*weights.get(n);
div = Math.sqrt(div);
//normalize
for(Neuron n: inputs)
updateWeight(n,weights.get(n)/div);
}
```