DYNAMIC SUBSET SELECTION APPLIED TO SELF-ORGANIZING MAPS

by

Leigh Wetmore

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University

Halifax, Nova Scotia
April, 2004

DALHOUSIE UNIVERSITY
FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled **"Dynamic Subset Selection Applied to Self-Organizing Maps"** by **Leigh Wetmore** in partial fulfillment of the requirements for the degree of **Master of Computer Science**.

Dated: _____

Supervisors: _____

Dr. Malcolm I. Heywood

_____

Dr. Nur Zincir-Heywood

Readers: _____

Dr. Raza Abidi

_____

Dr. Murray Heggie

DALHOUSIE UNIVERSITY

Date: _____

Author:	**Leigh Wetmore**
Title:	**Dynamic Subset Selection Applied to Self-Organizing Maps**
Faculty:	**Computer Science**
Degree:	**Master of Computer Science**
Convocation:	**May** Year: **2004**

Contents

List of Figures

List of Tables

Abstract

The self-organizing map (SOM) is an unsupervised learning algorithm that attempts to form a compact representation of a data set via a set of prototype vectors that exist in the same space. Dynamic subset selection (DSS) is a genetic programming (GP) based method of selecting the particularly difficult-to-learn patterns from a data set, where difficulty is a GP-specific measure. In this work, the dynamic subset selection self-organizing map (DSSSOM) is presented. It is the application of DSS to the SOM, with modifications to both. It is able to handle very large data sets, and has a DSS-based built-in stopping mechanism. The performance of the new algorithm is measured over five data sets via an original implementation, and compared to the SOM and other learning algorithms. Results show that the DSSSOM achieves a performance on par with the SOM with training time reduced by a factor of up to nearly five hundred.

Acknowledgments

This work was conducted while the author was funded by an NSERC graduate award. The author would like to especially thank Dr. Malcolm Heywood and Dr. Nur Zincir-Heywood for their invaluable supervision during this work. He would also like to thank Peter Lichodzijewski for the discussions on the many technical issues that cropped up. Finally, the author would like to thank his parents, Gordon and Susan Wetmore, for their encouragement and support.

1 Introduction

The dynamic subset selection self-organizing map (DSSSOM) is the application of dynamic subset selection (DSS) to the self-organizing map (SOM). The result is a new learning algorithm that is much faster than the SOM, and that performs at an equal level. The correct operation of the algorithm is determined via experiments that have been conducted on five distinct data sets.

The motivation for work on the DSSSOM stems from the continual need for learning algorithms that perform better and operate more quickly than existing algorithms. In particular, a motivating situation was encountered within the context of intrusion detection. There, the network devices on which intrusion detection was to take place were found to have insufficient computational ability to support an SOM-based intrusion detection system. This need ultimately led to the formation of the DSSSOM idea. It was during the development of the DSSSOM idea that it was recognized that the algorithm had the potential to replace the SOM in many applications outside of intrusion detection. The motivation for this work then became the enormous potential of the new learning algorithm: much faster learning coupled with a performance that is no worse than that of the SOM.

The SOM is an unsupervised learning algorithm that attempts to build a compact representation of a data space represented by the individual examples, or patterns, found in a given input data set. This representation is achieved by the ideal positioning of the SOM's prototype vectors, which exist in the same space as the input data. Traditionally, the entire data set is learned by the SOM in an iterative, sequential fashion. Ideally, groups of patterns in the data set represent regions of the input space that are linearly-separable from one another. Most data sets are unbalanced, that is, there is considerable variation in the sizes of the groups of patterns. The SOM is in essence a clustering algorithm, and as such, the allocation of resource to a particular group is dependent on its size. Thus, smaller groups often suffer from poor representation, and thus can be considered difficult to represent.

DSS effectively evens out the resource allocated to the different separable groups, but more specifically, it assigns more resource to difficult patterns. The input to the

algorithm is a data set for which each pattern has been labeled with a difficulty and an age. It then selects a subset of the data consisting of the oldest and most difficult patterns. The concepts of difficulty and age are defined by the learning algorithm to which DSS is applied. In the case of the SOM, pattern difficulty is defined in this work to be the distance from the pattern to the nearest SOM prototype vector. Age is defined to be the number of subset selections that have taken place since the pattern has actually been selected to the subset. Within the iterative operation of the SOM, instead of processing the entire data set each iteration, only the selected subset is processed. A pattern's difficulty refers to the difficulty with which the SOM is having forming a representation of it. Thus, the more difficult a pattern, the more often it is seen, and the more likely it is that a good representation of it will ultimately be achieved. The purpose of pattern age is to prevent pattern difficulty from completely dominating pattern selection; all patterns should be seen occasionally, no matter how low their difficulty.

The nature of the interaction between DSS and the SOM makes it impossible to simply place an implementation of DSS on top of an implementation of the SOM for use. The DSSSOM is therefore an original implementation of modified versions of both algorithms. DSS is modified for use with the SOM and large data sets. The SOM is modified to take advantage of a new training stopping condition that is based on pattern difficulty. The original implementation includes the traditional SOM algorithm for comparison purposes.

The enormous potential of the DSSSOM is explored via experiments conducted on five data sets: the adult data set, the forest covertype data set, NASA's space shuttle data set, the complete 41-feature KDD-99 data set, and a six-feature version of the KDD-99 data set. In each original experiment, predictive accuracies and training times are recorded for single DSSSOMs and SOMs, as well as for DSSSOM hierarchies. Predictive accuracies are measured over both seen (training) and unseen (testing) data sets. The predictive accuracies and training times, where available, of other learning algorithms are reported. The experimental objectives are four-fold: to show that the DSSSOM performs no worse, as measured by predictive accuracy over a labeled data set, than the traditional SOM; to show that the DSSSOM requires less training

time than the traditional SOM; to demonstrate that the DSSSOM achieves good predictive accuracy when compared to other (comparable) learning algorithms; and to demonstrate the generalization ability of the DSSSOM, that is, its ability to carry a good predictive accuracy over to an unseen data set after training.

The contribution of this work is the DSSSOM algorithm. In particular, it is an algorithm that

- learns, or trains, faster than the traditional SOM;
- performs at least as well as the traditional SOM;
- does not require experimentation to determine an ideal training length; and
- is able to learn much larger data sets than existing SOM implementations.

It is the firm belief of the author that the potential of the DSSSOM is such that it will one day be often used in place of the SOM.

2 Background

2.1 The Self-Organizing Map

The self-organizing map (SOM) is an unsupervised learning algorithm conceived by Kohonen in 1976 [9]. An SOM consists of an n-dimensional network lattice of nodes (called *neurons*). Most commonly, and in this work, two-dimensional hexagonal networks are used. A hexagonal network is one in which a neuron has at most six neighbours. Rectangular (four-neighbour) networks are also common.

To be learned is a set of m-dimensional vectors (or patterns) of real values, comprising the input (or training) data set. These vectors exist in the *input space*, while the n-dimensional positions of the neurons exist in the *output space*. Each neuron has associated with it an m-dimensional vector, called a *prototype vector*. The SOM is an example of *competitive learning*, as neurons compete to represent input patterns, with only one neuron winning each competition. As training (the learning of the training data set) proceeds, the neurons become tuned to various patterns (or groups of patterns), ultimately ordering themselves such that their locations are indicative of the features of the training data set [5]. Furthermore, the SOM is *topology preserving*, i.e., similar patterns are mapped to similar positions in the trained network. The topology of the network therefore is a visualization of the input data, no matter what its dimension.

The SOM algorithm has two phases: initialization and training. Initialization, the assigning of initial values to the prototype vectors, must take place before training can occur. This can easily be achieved by assigning them small random numbers. However, to give the SOM an added advantage, initialization is often based upon the training data. Here, the i^{th} value of a prototype vector is taken (or computed) from the set S of values comprising the i^{th} feature of the patterns in the training data set. Two popular initialization methods involve simply choosing values from S, or choosing values over the range of S. The latter method is used in this work.

Training may begin once initialization is complete. It involves the sequential presentation of the patterns in the training data set to the SOM, for each of which two processes occur: competition and adjustment. Let the prototype vectors and the pat-

tern to be learned be given by $\mathbf{p_i} = [p_i1, p_i2, ..., p_im]^T \in \mathbb{R}^m$ and $\mathbf{x} = [x_1, x_2, ..., x_m]^T \in \mathbb{R}^m$, respectively. Neuron competition is then based on the distances between the $\mathbf{p_i}$ and $\mathbf{x}$. In this work, the Euclidean distance function is used. The winning neuron w, called the best-matching unit (BMU), is the neuron that satisfies the following:

$$||\mathbf{x} - \mathbf{p_w}|| = \min_i ||\mathbf{x} - \mathbf{p_i}|| \tag{1}$$

The process of adjustment refers to the altering of the prototype vectors caused by the presentation of $\mathbf{x}$. The alteration is described by

$$\mathbf{p_i}(t+1) = \mathbf{p_i}(t) + h_{wi}(t)[\mathbf{x} - \mathbf{p_i}(t)] \tag{2}$$

where t is a discrete time during training and $h_{wi}(t)$ is called the *neighbourhood kernel function*. The kernel function satisfies two important properties: its highest value is at w, and its value decreases monotonically with increasing lateral distance [5]. This distance can be observed in figure 2.1. There, neurons are represented as hexagons in a two-dimensional hexagonal four-by-six network. The neuron labeled w is the current BMU; the other neurons are labeled with their lateral distances from w.

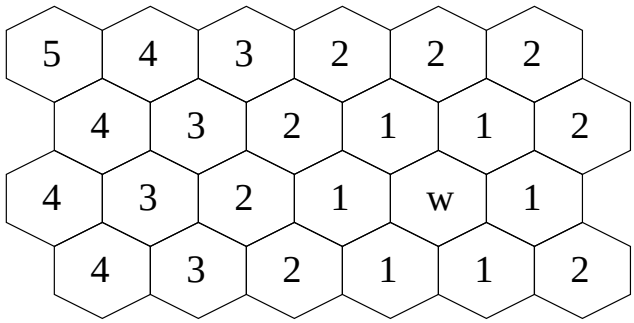


Figure 2.1: A 2-D hexagonal SOM network topology with lateral distances to BMU w.

The direction of prototype vector adjustment is toward x, in an attempt to better represent it. The kernel function is defined as

$$h_{wi}(t) = exp(-\frac{||l_w - l_i||^2}{2\sigma^2(t)}\alpha(t)) \tag{3}$$

where l_i is the position of neuron i in the network lattice, $\sigma(t)$ is the *neighbourhood*

radius function and $\alpha(t)$ is the *learning rate function*. These two functions are simple decay functions dependent on a time and an initial value. The decay is most often exponential, as it is in this work. The initial values are either implementation-dependent defaults, or values supplied by a human.

The discreet time t is normally measured in passes through the entire training data set. One such pass is called an *epoch*, and a number of epochs is a common SOM algorithm parameter used to define the training length. Training may also be stopped by monitoring the magnitude of the adjustments to the prototype vectors. Small adjustments are a good indication of a training data set that has been well-learned.

Training time is generally split into two phases: ordering and convergence. The ordering phase proceeds as described above. During convergence, however, the neighbourhood radius and learning rate are kept at fixed, low values. The duration of convergence is generally at least twice that of ordering. Its purpose is to fine-tune the network to the training patterns, and thus it is often referred to as fine-tuning (as it shall be henceforth here). Once fine-tuning has been completed, testing can begin. Testing involves feeding patterns into an SOM and identifying their BMUs. Evaluation depends on the application, though common evaluations are input space distance between a pattern and its BMU (and perhaps the other neurons as well), known as *quantization error*, or the comparison of pattern class labels with neuron labels. A neuron n can be labeled by testing an SOM with its training data set and observing the majority class of the patterns having BMU n (see section 2.3 below).

The SOM is essentially a clustering algorithm, with the prototype vectors representing the computed cluster centres. However, because the prototypes are connected together in a neural network, it can happen that, after training, some prototypes lie in the spaces between clusters. This is often seen as the biggest drawback of the SOM, as such prototypes do not represent the input data at all. Clearly, the ideal here is to have no prototypes at all in the spaces between clusters. However, suppose that the input space consists of several very large and several very small clusters. During training, the neural network will stretch to represent the dominant clusters, again leaving several prototypes in the regions between the large clusters. This time it is

desirable to have prototypes between the large clusters, and ideally they represent the small clusters. Unfortunately, the large clusters have too great an effect on such prototypes; they are yanked back and forth between the large clusters as the various patterns are processed during training, ultimately winding up potentially anywhere in between.

The SOM will perform well if the clusters in the given data set are linearly separable and of similar size. Non-separable data sets cannot be learned by the SOM and thus by the DSSSOM. It is in the situation of an *unbalanced* data set that a modification to the SOM algorithm is needed. For example, intrusion detection data sets often consist of one or more large clusters (normal traffic and perhaps common attacks) as well as many smaller clusters (uncommon attacks). It is vitally important that the smaller clusters be represented well to ensure a good rate of attack detection in a deployed SOM intrusion detection system. Active learning can be employed to do just this.

2.2 SOM Visualization

2.2.1 Hit Histograms

SOM visualizations are useful because from them information can be drawn that cannot be drawn (easily) from quantitative data. Of particular interest in this work are the hit histogram and the pattern label visualizations. Hit histograms are useful in demonstrating network ordering with respect to pattern class. To construct a series of hit histograms for an SOM, counts are maintained during testing at each neuron for each pattern class. The count for class c at neuron n is incremented if a pattern of class c has BMU n. Then, a histogram can be plotted for each class over all neurons. Ideally, there will be separation of different classes over the neurons. If there is not, the performance of the SOM will suffer. A hit histogram is a qualitative way of determining the performance of an SOM.

The most effective visualization of a hit histogram in this context can be seen in figure 2.2. There, neuron counts for a single class are represented by hexagon size for a $3x3$ network. Larger hexagons indicate larger counts of the class at that neuron.

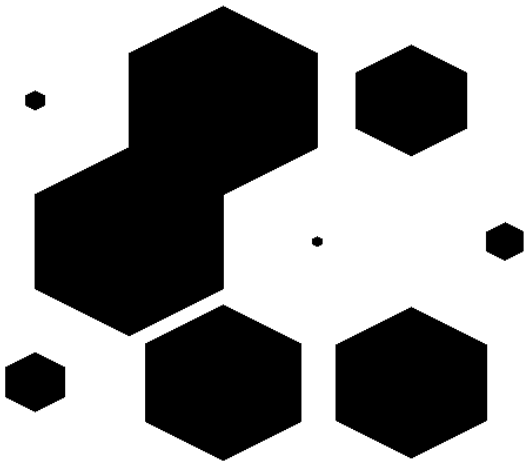


Figure 2.2: An example hit histogram.

2.3 Pattern Label Visualization

As was mentioned in section 2.1, a neuron can be labeled by testing an SOM with a data set and assigning the majority label of the patterns having it as BMU. If this is done for all neurons, a visualization can be formed. The simplest form of a pattern label visualization can be seen in figure 2.3, where labels have been affixed to each of the neurons. Labels can also be affixed to other types of visualizations, such as hit histograms.

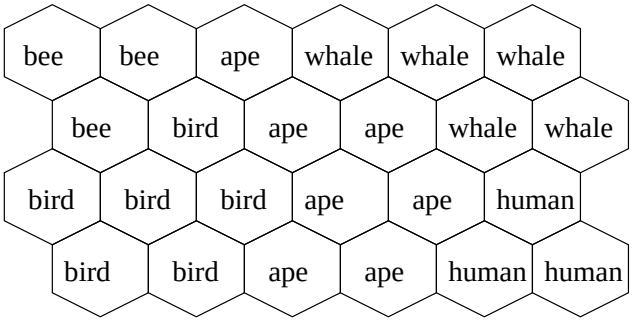


Figure 2.3: An example pattern label visualization.

2.4 Active Learning

Active learning [10] refers to the scenario in which a learner has some control over the data input to it. Presumably, this control involves selecting data that is expected to

yield the most positive desired result (e.g., a decreased error rate). Active learning is normally a two-phase process; the first phase involves selecting data into a training data set, while second involves training the learner on this data set. Though there are no known examples (other than those directly using dynamic subset selection, as described below), a one-phase version of active learning is to have the learner learn pattern subsets as they are built in an iterative fashion. Perhaps this version has not been used because it requires modification of the learning algorithm itself.

In the context of the SOM, the one-phase version of active learning involves having the SOM play some role in the selection of patterns from the training data set to be learned during the next epoch. Thus, returning to the example of an unbalanced data set above, smaller clusters can effectively made larger by increasing the frequency with which their patterns are selected. The result is a trained SOM that is better able to represent the data set as a whole. Dynamic subset selection applied to the SOM is a particular incarnation of active learning that is the focus of this work.

2.5 Dynamic Subset Selection

Dynamic subset selection (DSS) [4] was introduced by Gathercole and Ross in 1994 as an aid to genetic programming: it focuses a genetic program on difficult patterns, those that are often misclassified, while being careful not to completely avoid any patterns.

In the context of a genetic program (GP), a *generation* is a pass over the training data set followed by subsequent testing of the GP's *individuals* (independent classification units). With DSS applied, each generation involves two passes. The first pass is used to compute weights for the patterns in the training data set. The sum of the weights S is also computed. The weight of pattern i at generation g is defined as

$$W_i(g) = D_i(g)^d + A_i(g)^a, \quad D_i(0) = A_i(0) = 0 \tag{4}$$

where D and A are the difficulty and age of pattern i, and d and a are customizable exponents. During the second pass, each pattern i is assigned a probability P_i of

being selected into the dynamic subset, as described by

$$P_i(g) = \frac{W_i(g) * T}{S} \tag{5}$$

where T is the desired size of the resulting subset. Patterns selected into the subset have their difficulties and ages reset, while those not selected have their ages incremented by 1. During genetic program testing, the difficulty of a pattern is incremented by 1 each time it is misclassified by an individual. Note that the actual subset size is not fixed, and in fact it averages slightly above T. Gathercole and Ross felt that this might improve GP performance. They also point out that the selection of T, d, and a is arbitrary and depends on the properties of the training data set. Good values can only be determined through experimentation.

DSS requires access to the entire data set to which it is applied. This is impossible in the case of data sets too large to be stored in memory. The KDD-99 data sets used in this work are examples of such data sets. A common solution to this problem involves breaking a large data set up into subsets of contiguous patterns. Song does just this in his work on GP intrusion detection [16]. There, the subsets of patterns are called *blocks* and are of equal size (with the exception of the last block). DSS is applied to individual blocks read into memory; these blocks are selected at random with replacement.

Song also made two relevant changes to the DSS algorithm itself. The first eliminates selection of patterns by pattern weight, and replaces it with selection by difficulty *or* age alone, where there is a probability p $(1 - p)$ that the next pattern will be selected by difficulty (age). The second fixes subset size; patterns are selected until the subset is full. This version of the algorithm does not have a, d and T as parameters, and thus, the need for experimentation to determine their ideal values is eliminated.

Clearly DSS cannot be as effective on large data sets. It cannot control the selection of blocks, and therefore it cannot guarantee the selection of difficult patterns into its subsets. The following example serves to illustrate the effect of this shortcoming on this work. The KDD-99 10% data set contains 1925 patterns of the attack class *warezclient*. The *warezclient* patterns all occur in a section of the data set smaller

than two percent of its entire length. With a block size of 5000 patterns (as is used in this work), the patterns occur in only two blocks. Since *warezclient* patterns make up only 0.4 percent of the data set, DSS is required to help the SOM focus on the patterns. However, random block selection does not permit DSS to do this; only 1 in 50 blocks contain *warezclient* patterns, and so only two percent of the time will DSS be able to focus the SOM on *warezclient* patterns. Therefore, random block selection is not desirable; blocks containing uncommon (and thus difficult) patterns must be selected more often.

DSS in general is not directly applicable to other learning algorithms because pattern difficulty is computed using a GP-specific measure (the number of individuals incorrectly classifying the pattern). Of interest in this work is its applicability to the SOM. Since feedback must be obtained from the learning algorithm in order to compute pattern difficulty, and thus form the subset on which subsequent training takes place, DSS cannot be used with the traditional SOM in its current form.

2.6 Methods of Speeding Up Self-Organizing Map Training

The main benefit of applying DSS to the SOM is the resulting speed-up in training. Thus, other training speed-up methods are now presented. It shall be seen that DSS is quite unique.

2.6.1 Initialization by Clustering

A fast clustering algorithm can often be used to determine good initial values for the SOM prototype vectors. Su [17] presents an SOM training algorithm, hereafter referred to as the clustering SOM, or *CSOM*, that employs the k-means algorithm to do just this.

CSOM training proceeds in three stages. The first stage applies k-means to the training data set to obtain a set C of cluster centres of size N^2. The well-known k-means algorithm is given in [17], and shall not be reproduced here.

The second stage uses the discovered centres to initialize the prototype vectors of an NxN SOM, a procedure detailed in the subsequent listing. There, d refers to the Euclidean distance function, (x, y) refers to the prototype vector of the neuron at

output space location (x, y) in the neural network, and the selection of a centre implies its removal from C (and further consideration). Note that a rectangular topology is assumed.

1. For $i = 1$ to $\lceil \frac{N}{2} \rceil$,
   (a) Select u and v from C such that $d(u, v)$ is maximized, and assign them each to one of $(N - i + 1, i)$ and $(i, N - i + 1)$.
   (b) Select w such that $d(u, w) + d(v, w)$ is maximized, and assign it to (i, i).
   (c) Select x such that $d(u, x) + d(v, x) + d(w, x)$ is maximized, and assign it to $(N - i + 1, N - i + 1)$.
   (d) Select the $N - 2 * i$ centres with the $N - 2 * i$ largest combined distances to u and z, and assign them to the remaining unassigned prototypes in row i of the network, in order, with the centre closest to w adjacent to w and the centre furthest from w adjacent to v.
   (e) Repeat step 4 for row $N - i + 1$, and columns i and $N - i + 1$.
   (f) If $|C| = 0$, return.
   (g) If $|C| = 1$, assign the remaining centre to (i, i) and return.

The algorithm, while concise, is difficult to understand directly. Figures 2.4(a)-(f) describe the algorithm in a visual manner. The figures are snapshots of the neural network at different points in the algorithm. The neurons are represented by a grid of squares, with a black square indicating that the particular neuron has had its prototype initialized. Figures 2.4(a)-(e) are taken during the first iteration of the algorithm: (a) is taken following step 1(a), (b) is taken following step 1(b), and so on. Figure 2.4(f) is taken during the second iteration of the algorithm, after step 1(e). The figures show that the action of the algorithm is somewhat like the peeling of an onion: the outer layer or (in the two-dimensional case here) ring of neurons have their prototypes initialized first, followed by the next outermost ring, and so on, until the core is reached and all prototypes are initialized.

The third stage of the algorithm is traditional SOM training. Su claims that the CSOM has the ability to achieve an improved representation of the training data

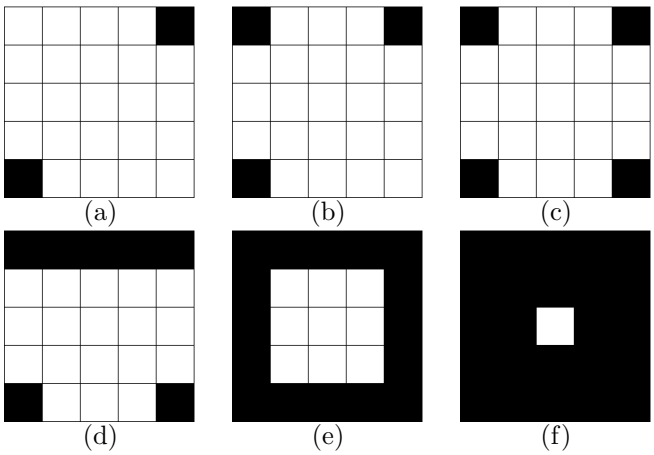


Figure 2.4: The operation of Su's k-means self-organizing map in assigning cluster centres to prototype vectors.

set in a shorter period of time. While a reasonable assumption, he neglected to run any experiments for which training time was measured. The experiments that were run focused on demonstrating an improved data representation. Three simple data sets were used: an artificial data set of two-dimensional points, the Iris data set, and the Animal data set. On each of these were trained three versions of the SOM algorithm: an SOM with random initialization, an SOM initialized using the data set, and the CSOM. Patten label visualizations (described in section 2.3) were used to report the results. From these visualizations, the CSOM appears to achieve greater separation over the pattern labels (classes) for the two-dimensional and Animal data sets, though this cannot be confirmed for certain without quantitative measures. The three algorithms yield nearly identical results in the case of the Iris data set.

There are several drawbacks to the CSOM. First, the neural network in use must be of square (NxN) dimension. Second, the network topology must be square (where each neuron has four neighbours). The cluster centre assignment listing above could however be modified for use with a hexagonal topology. Third, the CSOM focuses even more (than the SOM) on regions of high density in the input space, and even less on regions of low density. While this is desirable in some situations, it is undesirable in others. For example, in intrusion detection, entire classes of attack patterns often lie in regions of low density (there are very few patterns in these classes), and therefore are ignored by the CSOM. The resulting CSOM will not be able to detect such future

attacks.

2.6.2 Shortcut Winner Search

Kohonen proposes three SOM speed-up methods in [9], the first of which is the shortcut winner search. The general idea is to use the BMUs computed (and stored) at the previous SOM iteration to predict the BMUs at the current iteration. This scenario is only valid during fine-tuning, however, as during ordering the SOM can be quite unstable (leading to poor predictions).

Specifically, at iteration i during fine-tuning, each pattern p in the training data set is processed and its BMU $b_{p,i}$ is computed and stored. Then at iteration $i+1$ the BMU of p, $b_{p,i+1}$, is computed as follows.

1. Form the subset N_{i+1} of neurons consisting of $b_{p,i}$ and its immediate neighbours.
2. Compute the BMU $b_{N_{i+1}}$ for p over N_{i+1}, in the traditional manner.
3. If $b_{N_{i+1}} = b_{p,i}$, then $b_{p,i+1} = b_{p,i}$.
4. Otherwise, let $b_{p,i} = b_{N_{i+1}}$ and return to step 1.

Note that, if step 4 is reached, the computation of $b_{N_{i+2}}$ in iteration $i+2$ is greatly simplified. As $b_{N_{i+1}}$ and $b_{N_{i+2}}$ are neighbours, $N_{i+1} \cap N_{i+2} \neq \emptyset$. In particular, for a hexagonal topology, $|N_{i+1}| = 7$ and $|N_{i+1} \cap N_{i+2}| = 4$, and so only three distance computations need be undertaken. If all newly-computed distances are stored, even fewer computations may be necessary in future iterations.

Kohonen conducted an experiment to demonstrate the speed-up in SOM training when the above is applied. He measured the fine-tuning training time of two SOMs: a traditional SOM, and one with the above applied. Both had 768 neurons and were trained on a data set of size 9907 with pattern dimensionality 315. The sped-up SOM trained sixteen times faster (during fine-tuning) than did the traditional SOM. SOM performance on the given data set was not measured, however, and it is thus unclear as to whether good performance is sacrificed for speed.

2.6.3 Large SOM Estimation

This speed-up method, also proposed by Kohonen [9], applies mainly to situations in which a large number of neurons is required. The idea is quite simple: use the prototype vectors of a small, trained SOM to initialize the prototype vectors of a much larger SOM. If done properly, the large SOM will require much less training time. A requirement here is a smooth input space, so that the prototype vectors in the large SOM can be reasonably extrapolated from those of the small SOM. Unfortunately, no experiments that demonstrated increased training speed were reported.

2.6.4 SOM Smoothing

The final speed-up method proposed by Kohonen [9] is most applicable to situations in which there is insufficient training data to approximate the input space (as is required to build an accurate representation of it). Here, the desired SOM resolution (or representation accuracy) cannot be achieved. However, it may be possible to achieve the desired accuracy by stopping SOM training before fine-tuning, and applying a smoothing method instead. Smoothing aims to reduce random variation in the prototype vectors caused by the lack of training data. In a two-dimensional setting, the effect of smoothing can be seen in figure 2.5. There, the prototype vectors are denoted by x's. After SOM smoothing, notice that they are moved in the direction of the drawn line (approximately), which is the desired representation of the input space.

2.6.5 Hardware Implementation

Another method of speeding up SOM training is to train using a specialized hardware implementation of the SOM algorithm. One such implementation is presented by Lightowler in [14]. There, a scalable, modular, parallel implementation is presented requiring only two changes to the traditional software algorithm: input space distance metric and parameter quantization. Manhattan distances are used in place of Euclidean distances in order to significantly reduce hardware implementation expense. Parameter quantization occurs in the adjustment of prototype weight vectors; the value of the learning rate is restricted to negative powers of two. This quantiza-

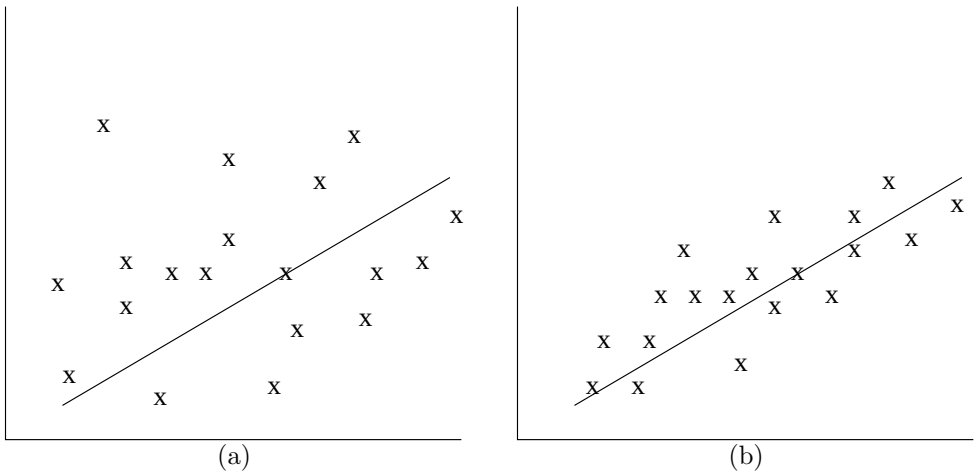


Figure 2.5: Two-dimensional prototype vectors (a) before SOM smoothing, and (b) after SOM smoothing.

tion then allows for a neuron to be implemented in hardware as a reduced instruction set computer (RISC) with three operations: addition and subtraction for computing Manhattan distances, determining neighbourhood sizes, and adjusting weight vectors; and the shift operation to adjust the learning rate during SOM training. The shift operation is far less expensive to implement than is the multiplication operation, which would be required if the learning rate was unrestricted.

An $8x8$ grid of neurons forms a *modular map*, or a module. Modules can be combined to make larger maps, though Lightowler suggests that steps be taken to minimize inter-module communication, such as restricting the output of a module to two eight-bit values (sufficient to indicate the position of a neuron). A module is restricted to learning input data of dimension sixteen or less. However, higher dimensional data can be learned by setting up a module hierarchy. Since the output dimension of a module is two, up to eight modules at a lower level may have their outputs fed as input to a single module at a higher level. Thus, a two-level hierarchy may accept input data of dimension at most 128, by having each of the eight first-level modules learn up to sixteen dimensions, and combining their outputs to form the second-level input. Even larger data sets may be processed by adding levels to the hierarchy.

Modular maps were trained on a human face recognition data set. The predictive accuracy (the proportion of the data set patterns whose labels were correctly pre-

dicted) was 96%, which compares favourably to the traditional SOM's accuracy of of 94%. Modular map training times were also reported to be lower, though no figures were given.

2.7 Data Sets

2.7.1 The Adult Data Set

Description. The adult data set, first used by Kohavi in [8], was extracted from the United States government's census bureau database, and contains 45222 labeled patterns. Each pattern has 14 features (or attributes) that describe a person, and one of two labels indicating a personal income of more than $50000 per year or otherwise. Traditionally, the first 30162 patterns form the training partition, while the remaining 15060 patterns form the testing partition. The data set is quite skewed, with only twenty-five percent of the patterns belonging to the $> \$50k$ class. The distribution of patterns in the two partitions can be seen in table A.1.

Goal. Given a pattern, to correctly predict the income of the described person. Predictive accuracy is measured over the testing partition.

Previous Results. Supplied with the adult data set are the classification accuracies attained by several existing learning algorithms on the testing partition. An accuracy is computed as the number of patterns correctly predicted divided by the total number of patterns seen. They are summarized in table 2.1.

Table 2.1: Previous adult testing partition classification accuracies.

Algorithm	*Accuracy*
NBTree	0.859
C4.5	0.845
Voted ID3	0.844
Naive Bayes	0.839
Nearest-Neighbour	0.796

2.7.2 The Forest Covertype Data Set

Description. The Forest Covertype data set was built by Blackard [1] out of the need for descriptive data of forested land to support decision making in ecosystem management. The entire data set contains 581012 labeled patterns, each of which

describes the forest cover of a specific piece of forest, belonging to one of seven classes, and using 54 measured attributes. The first 11340 patterns traditionally form the training partition. The next 3780 and 565892 patterns form the validation and testing partitions, respectively. The compositions of the three partitions, by pattern class, can be seen in table A.2. This partitioning of the covertype data set shall hereafter be referred to as *covertype-a*.

Goal. To correctly predict the cover type of the piece of forest described by a given pattern. Predictive accuracy is measured over the testing partition.

Previous Results. Only three classification results on the covertype data set are currently available, summarized in table 2.2. The results listed, in order, were attained by Blackard's neural network (backpropagation network), Cawley's SVM [2], and Rulequest's C5.0. The latter result was achieved with a training time of 679 seconds using a different data partitioning. There, half of the entire data set was used for training, while the other half was used for testing. The compositions of the two halves was not reported. This partitioning of the covertype data set shall hereafter be referred to as *covertype-b*.

Table 2.2: Previous Covertype testing partition classification accuracies.

Algorithm	*Partitioning*	*Accuracy*
Backpropagation Network	covertype-a	0.706
SVM	covertype-a	0.703
C5.0	covertype-b	0.941

2.7.3 The Shuttle Data Set

Description. The StatLog project [7] compared classification algorithms on large real-world problems. One of the data sets on which this comparison took place is the NASA-supplied shuttle control (or simply shuttle) data set. The data features describe a set of conditions during a space shuttle flight; pattern labels represent correct actions to take given the conditions.

There are 58000 labeled patterns, the first 43500 of which are commonly used as the training partition, with the remainder acting as the testing partition. Each pattern is described by nine features and one of seven labels. The data set is quite

skewed, with eighty percent of the patterns belonging to a single class. The pattern counts over the seven classes for both data sets are in table A.3.

Goal. To correctly predict the label of a given pattern. The testing partition is used to determine predictive accuracy.

Previous Results. The StatLog project obtained testing partition results using a number of learning algorithms. The most common and successful of these are listed in table 2.3.

Table 2.3: Previous shuttle data set classification accuracies.

Algorithm	*Accuracy*
NewID	0.100
CN2	0.100
C4.5	0.100
Nearest-Neighbour	0.996
Naive Bayes	0.954
Backpropagation Network	0.951

2.7.4 The KDD-99 Data Set

Description. The Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99) included the Third International Knowledge Discovery and Data Mining Tools Competition, for which an intrusion detection data set was constructed. Raw TCP dump data was recorded over seven weeks of simulated traffic on a local-area network designed to simulate a military network. This data was processed into five million labeled connection records (patterns) forming the training partition. Similarly, two weeks' worth of data were collected forming the testing partition. Two data sets used frequently in this work were derived from this data. The *10% KDD-99 data set* is a subset of the training partition of approximate size 500000. The *corrected KDD-99 data set* is a subset of the testing partition with approximate size 310000. The data sets are made up of connections (patterns) comprising five general class types: normal, denial of service (DoS), probe, root-to-local (R2L) and user-to-root (U2R). The latter four are attack types, each containing a number of individual attack classes. The testing (and corrected) partitions contain 17 attack classes not found in the training (and 10%) partitions. The exact sizes of the partitions, as well as their compositions by pattern class and type, is summarized in tables A.4-A.9.

Each KDD-99 pattern is described by 41 features and a class label (normal or one of the 39 attack types). The first nine features are referred to as the *basic features*. The latter 32 features are the *derived features*. Temporal information is encoded in the derived features, and thus the order of the patterns in the KDD-99 data sets need not be maintained.

Goal. To correctly predict whether a pattern is normal or an attack. Ideally a minimal false-positive rate and a maximal detection rate are achieved. A false positive is a normal pattern labeled as an (any) attack. A detection is an attack pattern labeled as an (any) attack. A secondary goal is to achieve these in as short a time as possible, a goal that applies to all time-critical prediction tasks. The corrected partition is traditionally used to measure success.

Previous Results. The following table summarizes results on the KDD-99 data set with the corrected partition used for testing. Both false positive and detection rates are included. The two co-winners of the Third International Knowledge Discovery and Data Mining Tools Competition, Pfahringer [15] and Levin [12], are listed first. Listed third is Kayacik's [6] result using an SOM. Listed fourth is Song's [16] genetic programming result. Eskin [3] obtained results using variants of three additional learning algorithms: the SVM, cluster-based estimation, and nearest neighbour; these are listed next. The origin of the final nearest-neighbour approach is unknown; it was one of the submissions to the KDD competition.

Table 2.4: Previous KDD-99 corrected data set detection and false positive rates.

Algorithm	*Detection Rate*	False Positive Rate
Pfahringer's C5 Ensemble	0.918	0.005
Levin's Kernel Miner	0.918	0.006
Kayacik's SOM	0.906	0.016
Song's GP	0.900	0.009
Eskin's SVM	0.980	0.100
Eskin's Estimation Clusterer	0.930	0.100
Eskin's Nearest-Neighbour	0.910	0.080
Other Nearest-Neighbour	0.909	0.005

2.7.5 The KDD-99 Six-Feature Data Set

Description. The six-feature version of the KDD-99 data set was first constructed and used by Kayacik [6]. As its name suggests, only the first six features in the data set are retained. Each of the features is used to construct a separate data set, with the idea being that a learning algorithm will be employed to learn each feature individually.

To regain temporal information lost in pruning the derived features of the patterns, a shift register is built for each of the six features, through which the values of the features are passed. Equidistant taps are placed along the register, and after each shift, the values at the taps form the next pattern in the feature-specific data set being constructed. The label of the original pattern, from which the most recent value to enter the register is taken, is used to label the newly-generated pattern. By processing the first six features of the 10% and corrected partitions in this way, six data sets are created for each that are used for training and testing purposes, respectively. The dimension of the patterns in the data sets depends on the number of shift register taps in use.

Previous Results. To this point, only Kayacik has run experiments on the KDD-99 six-feature data set. He achieved a false positive rate of 0.046 and a detection rate of 0.890 on the corrected data set, using an SOM hierarchy.

3 Dynamic Subset Selection Applied to Self-Organizing Maps

To make DSS applicable to the SOM, feedback from the SOM must be used to assign difficulties to the training patterns. The SOM prototype vectors are natural choices to provide this feedback. During training, prototype vectors are constantly being adjusted to better represent (be more similar to) the training patterns. A pattern is therefore represented well if it is similar (close, in the input space) to one or more prototype vectors. A pattern not being represented well is dissimilar to all prototype vectors, and can therefore be considered *difficult*. The dynamic subset selection self-organizing map (DSSSOM) defines the difficulty d of pattern $\mathbf{x}$ selected to a subset for the v^{th} time as

$$d(v) = \alpha||\mathbf{x} - \mathbf{p}_i|| + (1 - \alpha)d(v - 1) \tag{6}$$

where α is the current learning rate, and $\mathbf{p}_i$ is the prototype vector of the BMU of $\mathbf{x}$. Thus, the current difficulty is dependent upon the distance from $\mathbf{x}$ to $\mathbf{p}_i$ and $\mathbf{x}$'s previous difficulty, with more weight going to the latter as training proceeds.

3.1 Block Selection for Large Data Sets

To solve the problem discussed in section 2.5 concerning large data sets, blocks of data for learning must be selected in a non-random fashion. Fortunately the DSS concept of pattern difficulty allows for a simple solution: use the difficulties of the patterns within a block to compute a *block difficulty*. The simplest way of computing a block's difficulty is to compute the mean of the difficulties of its patterns. Another possibility is to use the median pattern difficulty. However, using means or medians alone can be misleading for different data distributions. In this work, a block's difficulty is defined as the difference in the mean and median difficulties of its patterns.

As with DSS, selection solely by difficulty is not a desirable property. To ensure that there is a very high probability of all blocks being seen during training, block age is kept track of. Selection of a block is then identical to the selection of a pattern into a DSS subset (see section 2.5).

3.2 A New Stopping Condition

The traditional SOM is normally trained until a specified number of epochs has been reached. Alternatively, the error on a validation data set can be used to stop training. The previous approach requires experimentation to determine the ideal number of epochs, while the latter is computationally expensive in implementation. In an attempt to solve both problems, the DSSSOM uses a stopping condition based on block difficulty that neither requires a specified training length, nor a validation set requiring expensive computation. The idea behind the stopping condition is that block difficulties are good indicators as to how well a data set has been learned. The average block difficulty can be expected to decline fairly sharply at first, and then level out as fewer patterns remain difficult. Training can be stopped when this leveling out occurs.

There are two problems to be solved before the described stopping condition can be used in practice: the lack of a specified training duration needed to compute the learning rate and neighbourhood radius, and the lack of a guarantee that training will stop. The learning rate and radius are traditionally computed using an exponential decay function. Such a function takes as arguments the current training time as well as the training duration. Fortunately, preliminary experimentation determined that the decay of the average block difficulty is roughly exponential. Thus, a linear mapping from the difficulty can be used to compute the learning rate and radius. Even better, a linear mapping means that it is possible for the learning rate and radius to *increase*, if the SOM's representation of the input patterns deteriorates during training. This allows the SOM to more quickly adapt and return to a state of better representation. SOM ordering is stopped when the learning rate (or radius) crosses a specified threshold. The learning rate is the more suitable choice as its value is a floating-point number. The chosen threshold is thus the fine-tuning learning rate. Once reached, SOM fine-tuning takes place for twice as many epochs as were spent on ordering. Note that in the context of the DSSSOM, an epoch is one block selection.

To solve the second problem, a mechanism needs to be in place to detect the leveling out of the average block difficulty (before the fine-tuning learning rate is reached). Indeed, there are difficult data sets for which the average block difficulty will

drop very little. One solution is to continually raise the fine-tuning learning rate (each change requires that training be restarted) until training successfully stops. However, in addition to being time-consuming, this solution will degrade SOM performance as the effectiveness of fine-tuning is reduced with higher fine-tuning learning rate values. An equally simple solution involves entering fine-tuning once the difficulty has leveled out. However, this could mean a sudden and very large drop in the learning rate and the radius, which again will have a negative effect on SOM performance. The two-pronged solution used in this work, although slightly more complex, avoids these large drops and does not require training restarts. The first idea is to reduce the learning rate and radius by a small amount each time the average block difficulty levels out (it will happen more than once if the data set is particularly difficult). Whereas during normal training the values of the learning rate and radius may increase, here maxima for both must be enforced. Otherwise, the SOM could return to its pre-reduction state. The second idea is to recompute the average block difficulty to learning rate and radius mapping when the average block difficulty reaches a new maximum. This will make it easier for the fine-tuning learning rate to be reached in the future.

In determining whether the average block difficulty has leveled out, a history of past average block difficulty values is kept. Leveling out is said to have occurred only if the average of the values is lower than the current value.

3.3 The Algorithm

The DSSSOM algorithm is concisely given here. Given for training is a large data set T containing t patterns. A block size of b is in use, and for each selection of a block B, n iterations of DSS are performed, each of which results in the selection of a subset S of size s. Recall the definitions of pattern and block difficulty and age. The current SOM learning rate is lr, and the fine-tuning learning rate is lr_{ft}. The current neighbourhood radius is r. There is a probability p_b (1 - p_b) of a block being selected in proportion to the block difficulties (ages), and a probability p_p (1 - p_p) of a pattern being selected in proportion to the pattern difficulties (ages). The current linear mapping from the average block difficulty to the learning rate and radius is m.

1. Initialize the difficulties and ages of the patterns in T by performing several

epochs of traditional SOM training on T.

2. Initialize the block ages. Using the pattern difficulties, initialize the block difficulties.

3. While $lr > lr_{ft}$,

   (a) Compute a random number r_b in $[0, 1)$.

   (b) If $r_b \leq p_b$, select a block B with probability in proportion to the block difficulties. Otherwise, select B by block age.

   (c) Read B in from disk.

   (d) For n iterations,

      i. Perform DSS on B. For s iterations,

         A. Compute a random number r_p in $[0, 1)$.

         B. If $r_p \leq p_p$, select a pattern p from B with probability in proportion to the difficulties of the patterns in B. Otherwise, select p by pattern age.

         C. Add p to S and remove it from further consideration.

      ii. Increment the ages of all patterns in B not selected to S.

      iii. For each pattern p in S,

         A. Train the SOM on p.

         B. Update the difficulty of p as per equation 6, and reset its age to 1.

   (e) Re-compute the difficulty of B given the updated pattern difficulties. Increment the ages of all blocks other than B.

   (f) If the average block difficulty has reached a new maximum, recompute m.

   (g) Use m together with the average block difficulty to update lr and r.

3.4 The Implementation

As the application of DSS to the SOM requires that changes be made to both, an original implementation was undertaken. It allows for the use of both the DSSSOM and the traditional SOM algorithms. It was used to run all original DSSSOM and SOM experiments in this work.

3.5 Properties of the DSSSOM

The testing partition accuracy of a given pattern class is largely determined by three factors: the content of that class in the training partition, the separability of the class from the other classes, and the number of patterns in the class. First, if the patterns of a class in the training partition are not representative of that class' patterns in the entire data set, the representation formed by the SOM will not be complete, leading to poor accuracy. Second, it is impossible for the SOM to form an accurate representation of a class that is not linearly separable from the other classes. Third, if there are too few patterns of a particular class in the training partition, those patterns are seen by the SOM algorithm too infrequently to have a lasting effect on the prototype vectors. The application of DSS to the SOM relieves this latter problem (to a degree that is controlled by the user) by effectively enlarging small classes via the concept of pattern difficulty. The other two problems cannot be alleviated without changing the SOM algorithm itself; in fact, DSS hampers the second problem by producing pattern difficulties that are deceivingly low. Two linearly non-separable patterns may be separated by a nonlinear boundary; this does not imply that the patterns are very different from one another; they may in fact be very similar (or near to) one another. Completely non-separable patterns may be even more similar.

4 Results

4.1 Experimental Setup

All experiments were run on a server running Mac OS X, with two 1.33 GHz G4 processors and 2 GB of RAM. Below, the experiments are divided by data set. The data sets in use have all been partitioned into training and test partitions (and possibly validation partitions as well). Of interest here are three general measures: training time, test partition accuracy, and training partition predictive accuracy. Training time refers to the amount of processor time spent completely training the architecture on which testing took place. Test and training partition definitions and characteristics were discussed in section 2.7.

Within each data set, the experiments are further divided into two types: DSSSOM-SOM comparison experiments, and DSSSOM predictive accuracy experiments. The former type of experiment aims to show that the DSSSOM achieves predictive accuracies that are on par with SOM predictive accuracies, but in less training time. The latter type of experiment aims to achieve the best DSSSOM predictive accuracy possible. The DSSSOM-SOM comparison experiments train single DSSSOMs and SOMs for comparison; there is no need to construct hierarchies of either for this task. However, preliminary experimentation showed that DSSSOM hierarchies achieved better predictive accuracies than did single DSSSOMs in most cases; thus, hierarchies are utilized in trying to achieve the best predictive accuracies. The hierarchical architectures in use are based on those first devised by Lichodzijewski in [13] and then more recently and relevantly used by Kayacik in [6].

Two-level hierarchies are built for all data sets, with the exception of the KDD-99 six-feature data set. The first level of the two-level hierarchy is simply a single DSSSOM trained on the entire training partition in the usual fashion. The training partition is then used to test the built DSSSOM, during which counts for each pattern class are maintained at each neuron. The count for class c at neuron n is incremented when a pattern of class c has BMU n. Neurons having significant counts for multiple classes contribute to classification error, because of the way in which patterns are labeled (a neuron takes the label of the majority class). In order to reduce this error,

second-level DSSSOMs are trained, one per such neuron, on only the patterns having that neuron as BMU. The selection of neurons for which this is done is manual at present, though it could easily be automated. To obtain the overall predictive accuracy of the hierarchy, the testing partition is run through, and its pattern labels compared to those of their BMUs at either the first or second-level, the latter being reserved for patterns having a BMU for which a second level has been built.

The three-level hierarchy is designed specifically for use with the KDD-99 six-feature data set described in section 2.7.5. Recall that the six-feature data set actually consists of six individual data sets, one for each of the first six attributes in KDD-99. At the first level, one DSSSOM is trained for each of these six parts; effectively each DSSSOM learns a single attribute. After training has been completed, the k-means clustering algorithm is used to cluster the prototype vectors of the six DSSSOMs into six clusters each. The six prototypes nearest to the six cluster centres effectively become the new set of prototype vectors for the network. Then, the DSSSOM is tested with the training partition and the quantization errors, or the distances to the (six) prototype vectors, for each pattern, are stored in an output file. An output file is simply a matrix of size $px6$, where p is the number of patterns in the training partition. The second-level training data set is a combination of the six first-level output files: the x^{th} pattern is built by concatenating the x^{th} rows of each of the six output files, as well as a pattern label. The second and third levels of the three-level hierarchy then exactly correspond to the first and second levels of the two-level hierarchy, the only difference being in the data used to train the lower of the two levels. In the case of the two-level hierarchy, it is the original training partition. In the case of the three-level hierarchy, it is the first-level combined output file. It should be noted that the first two levels of the three-level hierarchy *are* constructed and tested for DSSSOM-SOM comparison, as comparison at the first level is not as meaningful here as it is in the case of a two-level hierarchy.

Training times were recorded for all (originally) constructed maps and map hierarchies, and test partition accuracies were measured. Training partition accuracies were measured over the constructed DSSSOMs, as an indication of generalization ability. They were not measured over the constructed SOMs, as the generalization ability of

the SOM is not in question here. Validation partition accuracy, where applicable, was not measured.

Several parameters were kept constant across all experiments. These parameters, the algorithm to which they are applicable (SOM, $DSSSOM$, or $BOTH$), and their respective values, are listed in table 4.1. In the table, the acronym ABD refers to

Table 4.1: Parameters kept constant over all experiments.

Parameter	*Algorithm*	*Value*
Initial learning rate	BOTH	0.3
Fine-tuning learning rate	BOTH	0.01
Fraction of total training time spent fine-tuning	BOTH	0.67
Minimum training length	DSSSOM	100 block selections
Prob. of block selection by difficulty	DSSSOM	0.9
Prob. of pattern selection by difficulty	DSSSOM	0.7
Size of ABD history	DSSSOM	100
Effect of ABD leveling out	DSSSOM	radius reduced by 1
Top-level network dimension	DSSSOM	10x10
Trials performed per parameter set	DSSSOM	20
Trials performed per parameter set	SOM	1

the average block difficulty. It should be noted that when the neighbourhood radius is reduced after the average block difficulty levels out, the learning rate is reduced correspondingly. Also, the top-level network dimension refers to the size of the neural networks trained at the highest level of a DSSSOM hierarchy. The sizes of the maps trained at the lower level(s) of a hierarchy were not fixed.

Only one trial was performed per unique set of parameters in the case of the traditional SOM, though many different training lengths were used. The SOM will arrive at a fairly consistent solution given a data set that it is able to learn well. The same, however, cannot be said for the DSSSOM with certainty. Therefore, multiple DSSSOM trials are run per unique set of parameters.

The DSSSOM stopping condition in use does not guarantee a minimum training length. This can be a problem for particularly easy data sets, where training may stop before the average difficulty reaches its minimum value. To prevent this from happening, a minimum ordering training length of 100 epochs is enforced for all DSSSOM experiments (100 ordering epochs implies 200 fine-tuning epochs, for a total of 300 epochs). This limit potentially lengthens training time unnecessarily for some easy data sets, but it also prevents the poor performance that can occur as a

result of insufficient training for other data sets.

In the following sections, the network dimension of a first-level SOM or DSSSOM shall be referred to simply as its *size*. Testing partition predictive accuracy and training partition predictive accuracy shall sometimes be referred to simply as *testing accuracy* and *training accuracy*, repsectively.

4.2 Block and Pattern Selection Experiments

Early experimentation on the DSSSOM focused on showing that the algorithm did indeed select difficult and old patterns and blocks with a greater frequency than the others. The KDD-99 10% partition serves to demonstrate this. The data set was first ordered by class type, with the types containing the most patterns appearing first. Thus, the patterns were ordered as follows: *DoS*, *normal*, *probe*, *R2L*, *U2R*. A block size of 5000 was selected, resulting in the following distribution (in order): 78 blocks of *DoS*, one block of *DoS* and *normal*, 18 blocks of *normal*, one block of *normal* and *probe*, and finally a block of *probe* and all of *R2L* and *U2R* (99 blocks total). A DSSSOM was trained on this ordered data set, with block selection counts maintained. The counts are visualized in figure 4.1 as a histogram. Clearly, the latter blocks are selected more often. This is because the SOM has difficulty learning their patterns given the overwhelming majority of *DoS* and *normal* patterns. DSS age prevents *normal* and *DoS* from being shut out completely.

A similar experiment was conducted to demonstrate weighted pattern selection. A subset of the KDD-99 10% partition was created containing (in order) 93 *DoS* patterns, 2 *probe* patterns, 2 *R2L* patterns, and 3 *U2R* patterns for 100 patterns total. The block size, DSS subset size, and the number of subset selections per block selection were set at 100, 10, and 50, respectively. Pattern selection was based on difficulty (age) with probability 0.7 (0.3). A DSSSOM was trained for 400 epochs over which pattern selection counts were computed. The individual selection frequencies of the patterns can be seen in figure 4.2. In the figure, it is easy to see that the latter 7 patterns are slightly favoured, as would be expected. There are, however, some favoured *DoS* patterns; this is to be expected, as there are differences in difficulty over the individual classes that make up the *DoS* class type. The differences between the

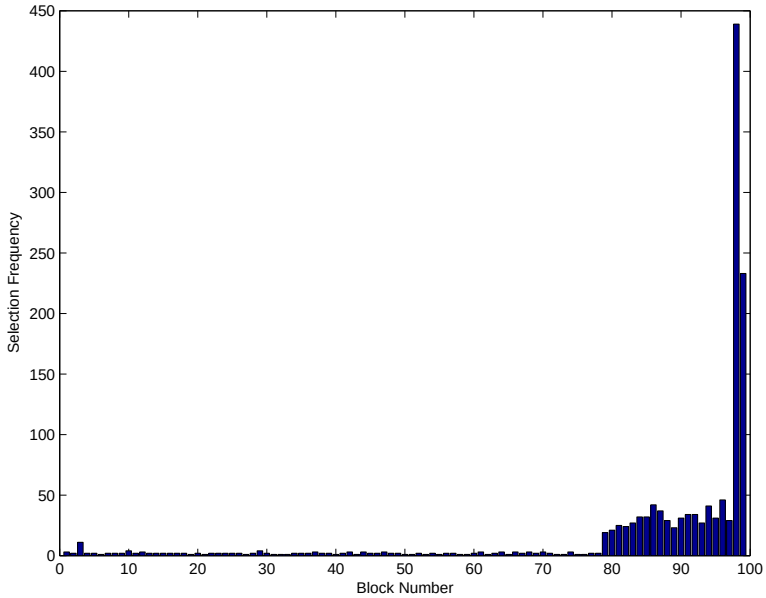


Figure 4.1: DSSSOM block selection frequency for the KDD 10% partition (ordered).

DoS and the other selection frequencies are smaller than one might expect because of the small data set size and the use of random selection to obtain the data set patterns.

4.3 Data Set Experiments

4.3.1 The Adult Data Set

An SOM was trained for each of six different training lengths and three different sizes. Twenty DSSSOM hierarchies were trained for each of three sizes. The DSSSOM parameters block size, DSS subset size, and number of subset selections per block selection were set at 500, 50 and 10, respectively. Table 4.2 presents the testing accuracies and training times for the SOM of each size with the best overall accuracy. It should be noted that the parameter set of each SOM trial is unique; the accuracies and training times of the other SOM trials are therefore made available in table B.1. Table 4.3 presents the testing accuracies and training times for the best DSSSOMs. Since the parameter sets were kept constant over the twenty DSSSOM trials for each size, additional DSSSOM results are omitted.

Second-level DSSSOMs were trained to form DSSSOM hierarchies for the three DSSSOMs in table 4.3. The training times and testing accuracies for the hierarchies

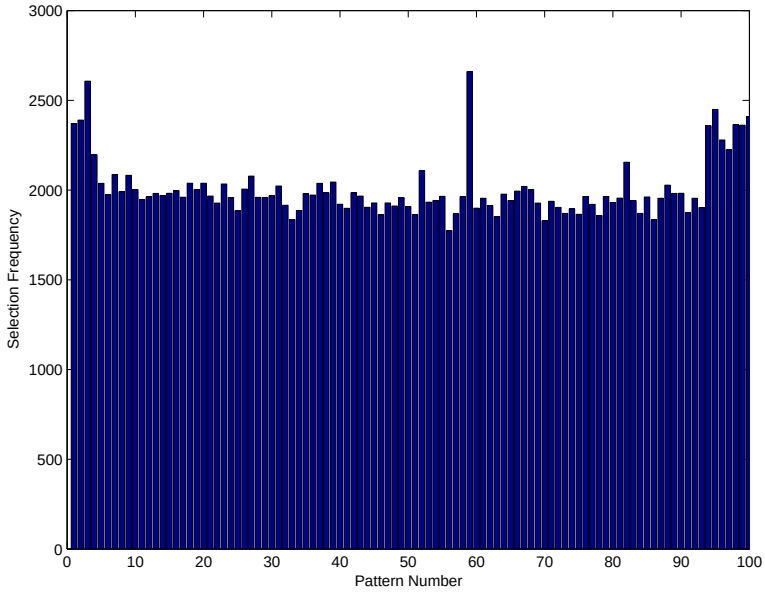


Figure 4.2: DSSSOM pattern selection frequency for a subset of the KDD-99 10% partition.

Table 4.2: SOM training times and predictive accuracies for the adult testing partition.

Size	*Training Length (epochs)*	*Accuracy*			*Time (s)*
		$\leq \$50k$	$> \$50k$	*Overall*	
10x10	1000	0.997	0.089	0.774	2509
8x8	4000	0.995	0.083	0.771	6890
6x6	50	0.999	0.033	0.762	55

Table 4.3: DSSSOM training times and predictive accuracies for the adult testing partition.

Size	*Accuracy*			*Time (s)*
	$\leq \$50k$	$> \$50k$	*Overall*	
10x10	0.996	0.103	0.776	141
8x8	0.995	0.087	0.772	96
6x6	0.997	0.071	0.770	73

can be seen in table 4.4. The corresponding training accuracies can be seen in table 4.5.

Hit histograms were generated for the first SOM in table 4.2 and for the first DSSSOM in table 4.3. They can be seen in figures 4.3 and 4.4, respectively.

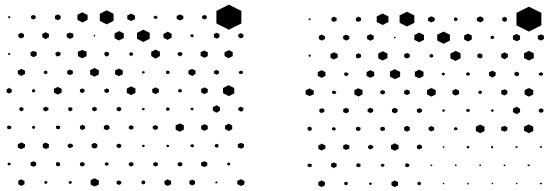
Looking at tables 4.2 and 4.3, it is clear that neither the SOM nor the DSSSOM can achieve a good testing accuracy. Though the largest DSSSOM achieves the highest accuracy, it is only marginally higher than the other DSSSOM and SOM accuracies.

Table 4.4: DSSSOM hierarchy training times and predictive accuracies for the adult testing partition.

	Accuracy			*Time*
Size	$\leq \$50k$	$> \$50k$	*Overall*	*(s)*
10x10	0.870	0.332	0.738	3648
8x8	0.918	0.291	0.764	2767
6x6	0.952	0.259	0.782	1587

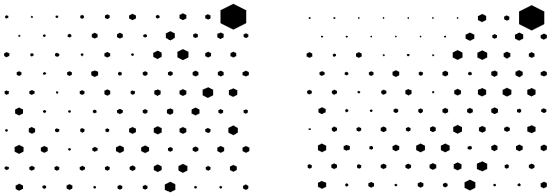
Table 4.5: DSSSOM hierarchy predictive accuracies for the adult training partition.

	Accuracy		
Size	$\leq \$50k$	$> \$50k$	*Overall*
10x10	0.952	0.433	0.823
8x8	0.962	0.361	0.812
6x6	0.973	0.295	0.804



(a) (b)

Figure 4.3: For an SOM trained on the adult data set, (a) the hit histogram for class $\leq \$50k$, and (b) the hit histogram for class $> \$50k$.



(a) (b)

Figure 4.4: For a DSSSOM trained on the adult data set, (a) the hit histogram for class $\leq \$50k$, and (b) the hit histogram for class $> \$50k$.

In fact, the SOM results are comparable at all training lengths down to ten epochs, and to the DSSSOM results; all however are poor in comparison to the results achieved by the other learning algorithms in table 2.1. Clearly the SOM cannot build a good enough representation of the adult data set to be able to differentiate between the two classes. Since the application of DSS makes little difference, the problem is likely not the imbalanced nature of the data set, but rather that the two classes are not easily separable. The hit histograms support this hypothesis, as both classes have

significant counts for the same neurons. Finally, although it is difficult to compare training times in this context unless good results are achieved, it is promising to note that no DSSSOM took longer to train than did an SOM trained for (a relatively short) 250 epochs.

Next, in comparing the accuracies of the individual DSSSOMs to the DSSSOM hierarchies (tables 4.3 and 4.4), it is clear that the construction of a hierarchy worsens the result. It shall be seen that this behaviour is limited to the adult data set. Building a second level had the effect of decreasing the $\leq \$50k$ accuracy while increasing the $> \$50k$ accuracy. The net result was a decrease in accuracy attributable to the large size of the $\leq \$50k$ class. It was noted that at nearly every neuron (regardless of hierarchical level), the distribution of the pattern classes closely resembled that of the adult data set as a whole. In an attempt to gain some separation over the two classes, several small experiments were conducted in which third levels were built on top of the second-level DSSSOMs. The third-level DSSSOMs were built in the same way as were the second-level DSSSOMs on top of the first level. Unfortunately no class separation was achieved. The best adult testing accuracy is therefore achieved by a single DSSSOM.

Finally, in comparing the testing and training partition accuracies of the DSSSOM hierarchies (tables 4.4 and 4.5), it is notable that there is a significant difference in both the class accuracies and the overall accuracy. As one would expect, the training partition accuracy is better, though still not good. There is little point in discussing the generalization ability of the DSSSOM here, as the data set must first be learned properly before good generalization can take place.

During DSSSOM training, the average block difficulty was consistently observed to drop from a very large value to a very small value, at which it leveled out. Furthermore, neither adult class was considered difficult relative to the other, as evidenced in figure 4.5. There, the two classes are observed to have had similar selection frequencies, despite the difference in size. A selection frequency for a particular class is computed by counting the number of times the patterns of the class are selected to DSS subsets, and dividing by the class size. The selection frequencies of small and/or difficult classes should be higher. Based on these observations alone, one would natu-

rally conclude that good representations of both classes have been formed. However, the observed training and testing accuracies, as well as the identical pattern distributions for the two classes (figure 4.4), strongly indicate that this is not the case. This contradiction can only be explained by poor linear separability. In such a scenario, it is possible for both classes to have very low pattern difficulties, as both classes may be very near to the same neurons. Predictive accuracy, however, will be very poor, as an SOM neuron is labeled by the majority class; the patterns of the other class, though having the neuron as BMU, have their labels predicted incorrectly. Figure 4.6 shows the pattern label visualization for the first DSSSOM in table 4.3. Notice that although the $> \$50k$ class has significant counts on many of the neurons (figure 4.4), the vast majority of those neurons are labeled as $\leq \$50k$.

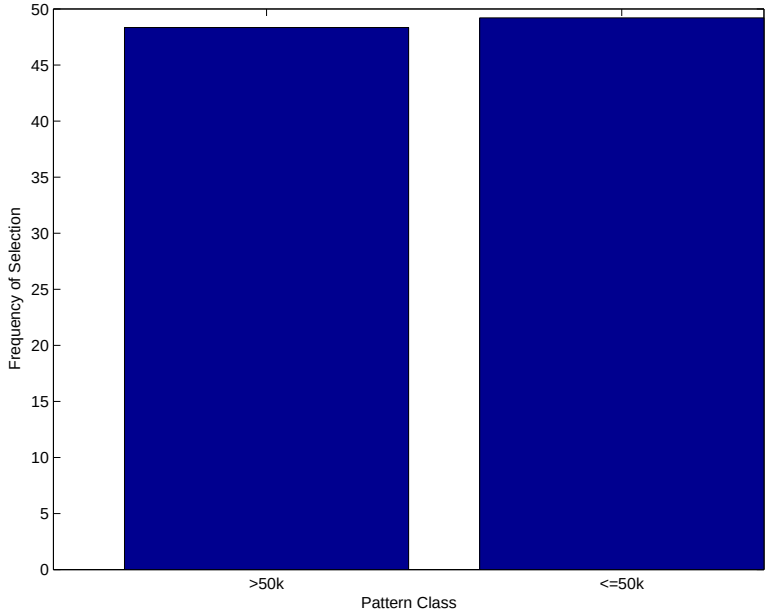


Figure 4.5: DSSSOM pattern selection frequency, by pattern class, for the adult data set.

4.3.2 The Covertype Data Set

The experiments performed on the adult data set were exactly repeated for the the covertype-a data set, and the results are presented in the same fashion.

As with the adult data set, the overall accuracies of both architectures on the covertype-a data set are quite poor (tables 4.6 and 4.7). The accuracies of the SOM

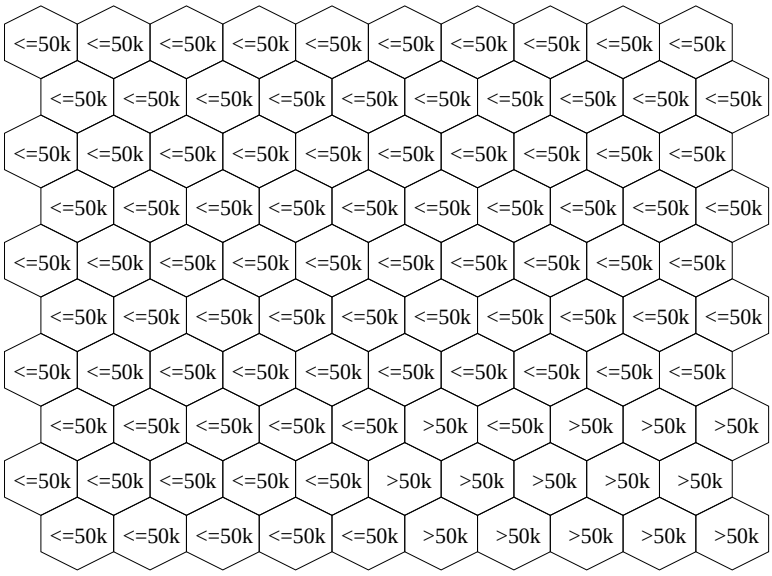


Figure 4.6: For a DSSSOM trained on the adult data set, the neuron label assignments.

are again inconsistent due to its inability to learn the data set. Evidence of this inability is easy to spot; the best accuracy achieved by a 10x10 SOM required a mere ten epochs (including fine-tuning) of training time, certainly not enough to ensure either proper ordering or fine-tuning, especially on a data set as large as covertype-a. The best SOM accuracies were better than the best DSSSOM accuracies, though it is impossible to tell whether this is a result of a weakness of the DSSSOM or of random chance. The DSSSOM accuracies were much more consistent, likely due to the enforced minimum training length of 300 epochs (total). In any case, the SOM algorithm cannot learn the covertype-a data set, and again a comparison of training times would have little meaning.

Table 4.6: SOM training times and predictive accuracies for the covertype-a testing partition.

	Tr.Len.	*Accuracy*								*Time*
Size	*(epochs)*	*S/F*	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*	*(s)*
10x10	10	0.371	0.231	0.241	0.709	0.526	0.370	0.696	0.306	24
8x8	50	0.396	0.206	0.414	0.671	0.446	0.193	0.538	0.303	76
6x6	250	0.280	0.081	0.306	0.719	0.396	0.295	0.536	0.194	233

Unlike the adult data set, the covertype-a data set enjoys a significant improvement in accuracy when a two-level DSSSOM hierarchy is used, as seen in table 4.8.

Table 4.7: DSSSOM training times and predictive accuracies for the covertype-a testing partition.

	Accuracy								*Time*
Size	*S/F*	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*	*(s)*
10x10	0.359	0.174	0.296	0.658	0.616	0.253	0.613	0.272	191
8x8	0.372	0.157	0.398	0.647	0.564	0.277	0.613	0.274	120
6x6	0.242	0.179	0.000	0.809	0.460	0.277	0.601	0.213	54

The training times are (not surprisingly) significantly longer, but the accuracies are more than twice as good. The improvement is not restricted to a single class, but rather spread evenly over the classes. The *lodgepole pine* class continues to have the lowest accuracy, and because of the large number of patterns in that class, the overall accuracy remains poor (relative to the other learning algorithms presented in table 2.2). Table 4.9 presents a confusion matrix corresponding to the first DSSSOM hierarchy in table 4.8. The value at position (x, y) in the table corresponds to the number of patterns of label x that have had their labels predicted by the hierarchy as y. It can be seen that the bulk of the overall error is attributable to the large number of *lodgepole pine* patterns misclassied as *spruce/fir*, and vice versa. The two classes are also often misclassifed as *aspen*, *douglas fir*, and *krummholz*.

Table 4.8: DSSSOM hierarchy training times and predictive accuracies for the covertype-a testing partition.

	Accuracy								*Time*
Size	*S/F*	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*	*(s)*
10x10	0.784	0.375	0.601	0.809	0.747	0.583	0.750	0.563	6130
8x8	0.748	0.366	0.602	0.860	0.815	0.577	0.778	0.547	3863
6x6	0.661	0.398	0.579	0.848	0.821	0.579	0.802	0.530	1908

Table 4.9: A DSSSOM hierarchy confusion matrix for the covertype-a testing partition.

	S/F	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*
S/F	164428	27446	766	0	4387	511	12142	209680
LPine	138817	105462	8968	176	18505	6021	3192	281141
PPine	5900	711	20201	2334	407	4038	3	33594
C/W	41	0	56	475	0	15	0	587
Asp	1199	384	192	0	5481	77	0	7333
DFir	2387	373	2293	1034	259	8861	0	15207
Krum	4293	278	0	0	15	0	13764	18350
Overall	317065	134654	32476	4019	29054	19523	29101	565892

The performance of a DSSSOM hierarchy on the training partition is presented in table 4.10. It was much better than was the performance on the testing partition. The biggest improvement comes by way of the *lodgepole pine* class, which accounts for the majority of the overall increase. Clearly the DSSSOM does not generalize well here, though this is to be expected in the case of a data set that has not been learned well to begin with.

Table 4.10: DSSSOM hierarchy predictive accuracies for the covertype-a training partition.

	Accuracy							
Size	*S/F*	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*
10x10	0.942	0.826	0.894	0.942	0.949	0.840	0.970	0.909
8x8	0.898	0.733	0.813	0.941	0.937	0.765	0.941	0.861
6x6	0.804	0.669	0.711	0.887	0.921	0.707	0.908	0.801

The histograms in figures 4.7 and 4.8 show some separation over the classes. Still, all of the histograms show significant counts at many neurons, though the distribution is not as even as it was for the adult data set. The DSSSOM histograms for the *spruce/fir*, *lodgepole pine* and *krummholz* classes are very similar, as are those of the *ponderosa pine*, *cottonwood/willow* and *douglas fir* classes. The *aspen* class is a mix of the two groups. The *spruce/fir*, *lodgepole pine* and *krummholz* group indeed accounts for most of the predictive error seen in table 4.9. Most of the remaining error is caused by misclassification amongst the three classes of the other group. The poor predictive accuracies achieved were caused by the DSSSOM's inability separate the classes within these groups. The pattern label visualization in figure 4.9 is an agglomerative form of the class-specific hit histograms, with the majority class labeling each neuron. It can be seen there that the classes within the groups tend to be near to one another, as they compete to occupy the same regions of the map.

Not surprisingly, given the discovered predictive accuracies, the DSSSOM pattern selection frequency chart in figure 4.10 shows that there is little variation in selection frequencies over the classes (much more variation shall be seen later). The DSSSOM therefore finds no class particularly difficult, though it does find the class on which the worst predictive accuracy was achieved to be the most difficult (*lodgepole pine*). Even though *lodgepole pine* is the largest class, it perceived by the DSSSOM to be

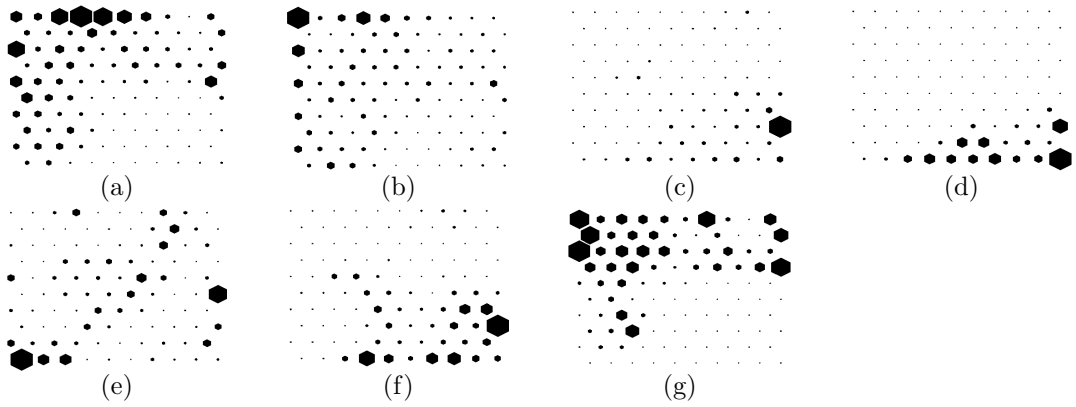


Figure 4.7: For an SOM trained on the covertype-a data set, hit histograms for the classes (a) *spruce/fir*, (b) *lodgepole pine*, (c) *ponderosa pine*, (d) *cottonwood/willow*, (e) *aspen*, (f) *douglas fir*, and (g) *krummholz*.

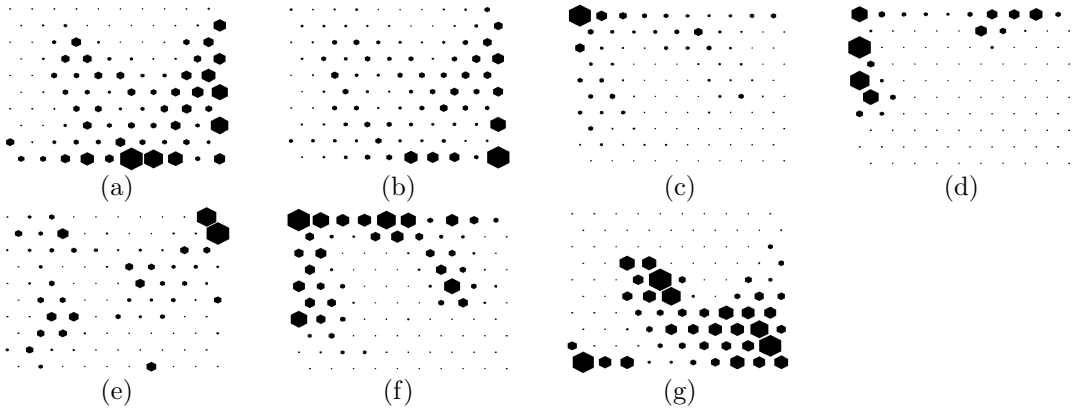


Figure 4.8: For a DSSSOM trained on the covertype-a data set, hit histograms for the classes (a) *spruce/fir*, (b) *lodgepole pine*, (c) *ponderosa pine*, (d) *cottonwood/willow*, (e) *aspen*, (f) *douglas fir*, and (g) *krummholz*.

difficult because separation could not be achieved from the *spruce/fir* and *krummholz* classes.

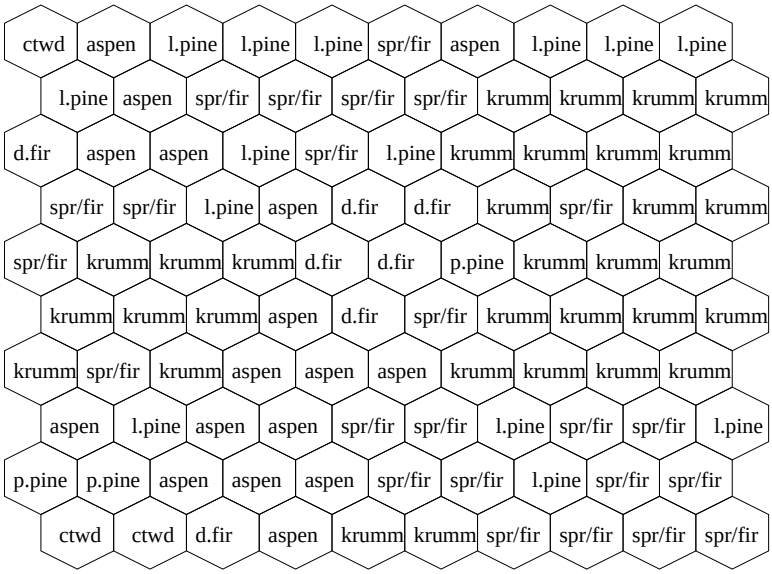


Figure 4.9: For a DSSSOM trained on the covertype-a data set, the neuron class labels.

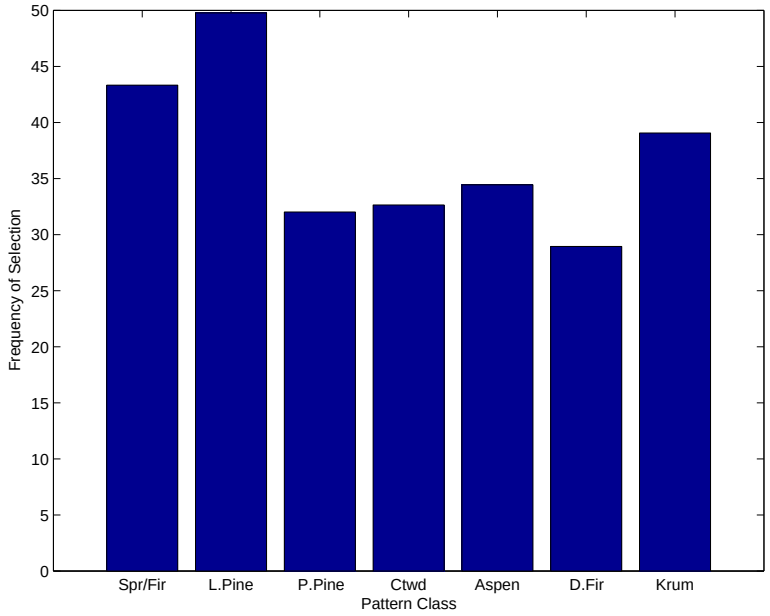


Figure 4.10: DSSSOM pattern selection frequency, by pattern class, for the covertype-a data set.

4.3.3 The Shuttle Data Set

The same experiments were again repeated for the shuttle data set. Tables 4.11 and 4.12 clearly show that the DSSSOM requires far less training time than does the SOM to achieve the best possible overall accuracy. Table B.3 presents the entire set of SOM results. There, it can be seen that the results are good even when the training length is severely restricted. However, it is not until the training length is restricted to fifty epochs or less that the DSSSOM and SOM training times become comparable. The histograms in figures 4.11 and 4.12 give good reasoning for the high accuracies. Unlike the adult and covertype data sets, the pattern classes in shuttle are easily separable. The separation is quite obvious in the figures. There is still some overlap, mostly in the larger classes (*rad flow*, *fpv open*, *high*), which prevents even higher accuracies from being achieved.

Table 4.11: SOM training times and predictive accuracies on the shuttle testing partition.

Size	*Tr.Len. (epochs)*	*Accuracy*								*Time (s)*
		RFlow	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Overall*	
10x10	1000	0.981	0.154	0.231	0.871	0.994	0.000	0.000	0.963	2826
8x8	4000	0.981	0.308	0.000	0.762	0.994	0.000	0.000	0.946	7525
6x6	1000	0.951	0.000	0.000	0.771	0.989	0.000	0.000	0.923	1150

Table 4.12: DSSSOM training times and predictive accuracies on the shuttle testing partition.

Size	*Accuracy*								*Time (s)*
	RFlow	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Overall*	
10x10	0.989	0.077	0.000	0.869	0.991	0.000	1.000	0.968	145
8x8	0.981	0.154	0.000	0.808	0.948	0.000	0.000	0.950	64
6x6	0.967	0.000	0.000	0.699	0.947	0.000	0.000	0.923	24

The pattern label visualization in figure 4.13 summarizes the DSSSOM hit histograms in figure 4.12. The *rad flow* class, having significant counts at many neurons, dominates the SOM landscape. The neurons labeled *high* clearly correspond to the most densely populated neurons of the *high* hit histogram. The largest concentrations of *bypass* and *bpv open* also correspond to labeled neurons. Notice that the *fpv open* and *bpv close* have no labeled neurons, explaining the zero accuracies in the first row of table 4.12. Notice also that the *fpv close* class has very poor accuracy because the

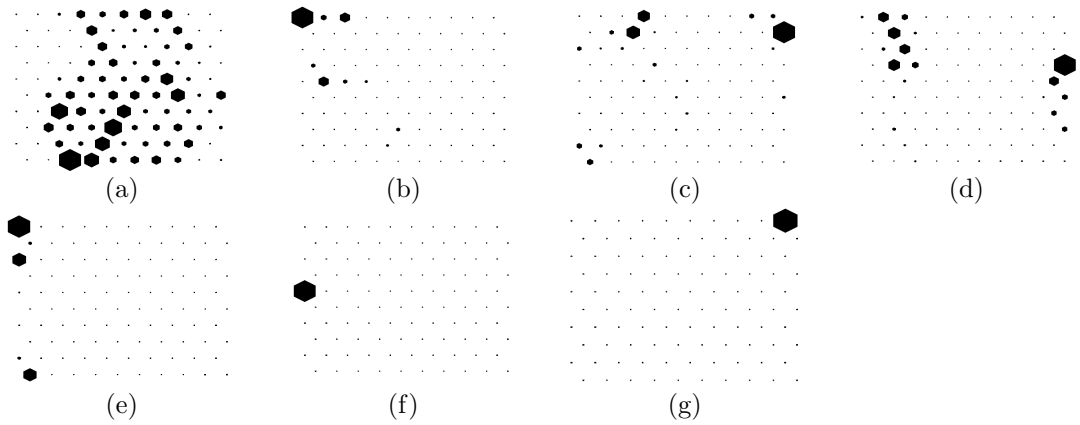


Figure 4.11: For an SOM trained on the shuttle data set, hit histograms for the classes (a) *rad flow*, (b) *fpv close*, (c) *fpv open*, (d) *high*, (e) *bypass*, (f) *bpv close*, and (g) *bpv open*.

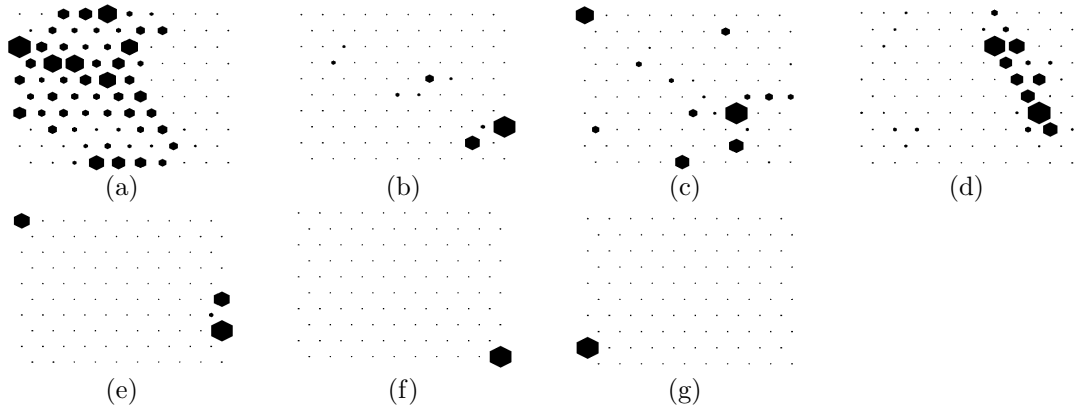


Figure 4.12: For a DSSSOM trained on the shuttle data set, hit histograms for the classes (a) *rad flow*, (b) *fpv close*, (c) *fpv open*, (d) *high*, (e) *bypass*, (f) *bpv close*, and (g) *bpv open*.

neurons for which its class count is highest are dominated by other classes, though it is difficult to see this given that the sizes of the hexagons from one histogram to the next are not comparable (a larger class size means that smaller hexagon sizes will represent higher counts). The smaller classes (*fpv close*, *fpv open*, *bpv close* and *bpv open*) do not affect the overall accuracy very much. The bulk of the predictive error comes from neurons with significant counts for two of the three large classes; the addition of a second layer of DSSSOMs achieves the desired separation for these neurons, as shall be seen.

The bar chart in figure 4.14 attempts to quantify the difficulty of the shuttle classes by presenting their pattern selection frequencies. All of the classes have comparable

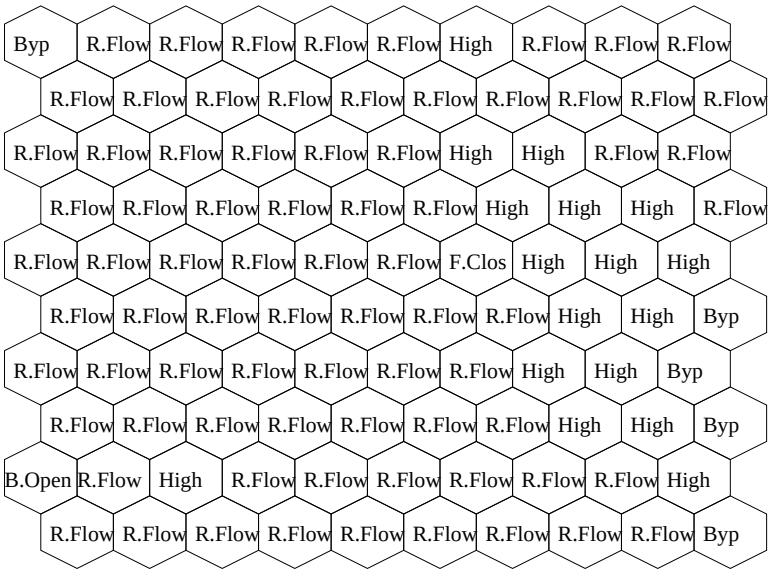


Figure 4.13: For a DSSSOM trained on the shuttle data set, the neuron class labels.

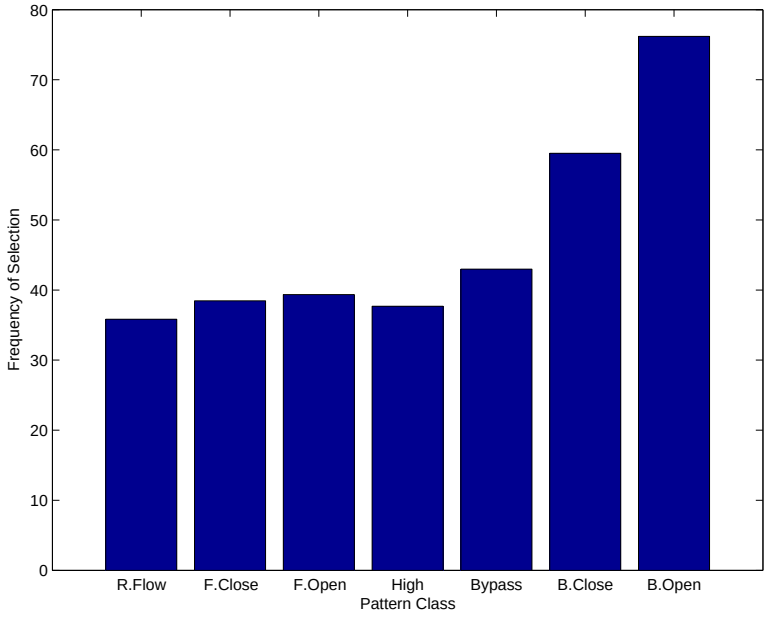


Figure 4.14: DSSSOM pattern selection frequency, by pattern class, for the shuttle data set.

frequencies, with the two smallest classes (*bpv close* and *bpv open*) having the highest frequencies; still, none of the small classes are selected often enough to overcome the much larger *rad flow*, *high* and *bypass* classes. Since the DSSSOM fails to see the

smaller classes as being particularly difficult, it is reasonable to assume that they are at least somewhat similar to the larger classes. It is also very possible that the DSSSOM stopping condition caused training to stop before the smaller classes had been learned; their relatively few patterns would contribute little to the average block difficulty measure used to stop training.

The addition of a second level of DSSSOMs improves the overall testing accuracies significantly, as shown in table 4.13. The largest percentage improvements come in the smaller classes, which were not well learned at the first level. However, the bulk of the improvement can be attributed to the three larger classes, as good separation between them was achieved at the second-level. The confusion matrix for the first DSSSOM hierarchy in table 4.13 is presented in table 4.14. There, it can be seen that most of the predictive error resulted from *high* being misclassified as *rad flow* and vice versa; furthermore, almost all of the remaining error was caused by the other classes being misclassified as one of the two. This is not surprising, given that large classes often have relatively small counts at neurons that are relatively large compared to smaller classes, resulting in misclassification.

Table 4.13: DSSSOM hierarchy training times and predictive accuracies on the shuttle testing partition.

	Accuracy								*Time*
Size	*RFlow*	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Overall*	*(s)*
10x10	0.999	0.308	0.615	0.975	0.991	0.500	0.000	0.993	797
8x8	0.997	0.385	0.385	0.968	0.986	0.500	0.000	0.990	693
6x6	0.999	0.538	0.769	0.983	0.994	0.250	0.500	0.995	876

Table 4.14: A DSSSOM hierarchy confusion matrix for the shuttle testing partition.

	RFlow	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Overall*
RFlow	11464	0	1	13	0	0	0	11478
FClose	5	4	0	4	0	0	0	13
FOpen	13	0	24	1	1	0	0	39
High	52	0	0	2101	2	0	0	2155
Byp	7	0	0	0	802	0	0	809
BClose	2	0	0	0	0	2	0	4
BOpen	2	0	0	0	0	0	0	2
Overall	11545	4	25	2119	805	2	0	14500

The overall predictive accuracies change little when the training partition is used

in testing. This is to be expected given the high testing accuracies. The improvement is spread quite evenly over the classes, though the percentage difference is again the highest for the smaller classes. The DSSSOM generalizes well in the cases of the larger classes, but not very well in the cases of the smaller classes. The smaller classes are either too similar to the larger classes, or simply too small. If the latter is true, the DSSSOM stopping method must be changed such that more weight is given to smaller classes during average block difficulty computation. This will have no effect, however, if the small classes are indeed too similar to (and thus not easily separable from) the larger ones.

Table 4.15: DSSSOM hierarchy predictive accuracies on the shuttle training partition.

	Accuracy							
Size	*RFlow*	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Overall*
10x10	1.000	0.757	0.644	0.986	0.996	1.000	1.000	0.996
8x8	0.998	0.811	0.591	0.976	0.995	1.000	0.909	0.993
6x6	0.998	0.784	0.735	0.990	0.996	1.000	0.909	0.996

The performance of DSSSOM on the shuttle data set is as good as any of the non-decision tree-based algorithms presented in table 2.3. With a training time of only about fifteen minutes for the best hierarchical DSSSOM result (comprised of twenty-five DSSSOMs), a very good predictor with overall accuracy 0.995 can be constructed in a very short time.

4.3.4 The KDD-99 41-Feature Data Set

It shall be seen that the KDD-99 data set is a good data set on which to demonstrate the abilities of the DSSSOM. It is neither too difficult nor too easy to learn, and the large size of the commonly used 10% and corrected partitions serves to illustrate the DSSSOM effect on training time. The pattern classes of the KDD-99 data set can be broken up into three partitionings: *normal* and *attack*; *normal*, *DoS*, *probe*, *U2R*, and *R2L* (the latter four are referred to as *attack types*); and of course the individual classes themselves. Experimental results are largely reported in terms of the second partitioning. A correct classification in the case of an attack type (e.g., *DoS*) is said to have taken place if the pattern in question is classified as *any* of the four attack

types. Of interest here is the ability to classify an attack as *attack* and normal traffic as *normal*; the correct prediction of attacks as particular attack types or individual classes is secondary, though the results in this respect are easily seen below.

Twenty two-level DSSSOM hierarchies of three different first-level sizes (10x10, 8x8 and 6x6) were trained on the 10% partition. The DSS block size, subset size, and number of subset selections were set at 5000, 250 and 20, respectively. Tables 4.16 and 4.17 present the false positive and detection rates of the best hierarchies of each size, on the 10% and corrected partitions, respectively. The accuracies are summarized by class type. A more complete version of the results is presented in tables B.4-B.9. There, individual class accuracies are reported for both the 10% and corrected partitions. For each class c, the three other classes most commonly predicted for the patterns of class c (the most common misclassifications of class c) are given. Table B.9 is a summarizing confusion matrix for the class types.

Table 4.16: DSSSOM hierarchy training times and predictive accuracies on the KDD-99 41-feature corrected partition.

	Accuracy							*Time*
Size	*Normal*	*DoS*	*Probe*	*R2L*	*U2R*	*FP*	*Det.*	*(s)*
10x10	0.995	0.972	0.704	0.008	0.000	0.005	0.904	10544
8x8	0.980	0.970	0.692	0.008	0.043	0.020	0.902	12697
6x6	0.995	0.970	0.709	0.008	0.029	0.005	0.903	12161

Table 4.17: DSSSOM hierarchy predictive accuracies on the KDD-99 41-feature 10% partition.

	Accuracy						
Size	*Normal*	*DoS*	*Probe*	*R2L*	*U2R*	*FP*	*Det.*
10x10	0.999	0.999	0.784	0.885	0.192	0.001	0.996
8x8	0.993	0.999	0.632	0.876	0.250	0.007	0.995
6x6	0.993	0.999	0.780	0.825	0.173	0.007	0.996

Table 4.16 shows that the DSSSOM compares nicely with Kayacik's SOM result in table 2.4. The detection rate is slightly lower (0.002), but the false positive rate is reduced by sixty-eight percent (0.011). In order to compare the training times of the two architectures, a two-level SOM hierarchy was trained on the KDD-99 10% partition. The training length and network size were set at 4000 and $6x6$, as they were in Kayacik's work. The measured training time was 33.4 hours, or about ten times as long as a two-level DSSSOM hierarchy of the same size that achieves comparable

accuracies.

The DSSSOM performs very well on the 10% partition, especially for size $10x10$. All attack types as well as *normal* experience improvement, with the biggest percentage improvement being *R2L*. The DSSSOM does not have any trouble learning the *normal* or *DoS* types. The other three types are, however, problematic. The *R2L* accuracy is fairly good in the case of the 10% partition, but it drops to virtually zero in the case of the corrected partition. The *U2R* type performs poorly over both partitions. The *probe* performance is mediocre in both cases.

To determine the approximate difficulty associated with each class type, a DSSSOM was trained and counts kept for the class labels as patterns were selected to DSS subsets for presentation to the SOM algorithm. In figure 4.15, each bar represents the count of a particular type (all four attack types and *normal*), divided by the number of patterns of that type, for the 10% partition. One would expect to see higher values for the smaller (and presumably more difficult) types, and this is very nearly the case. The frequency with which *normal* patterns are selected is higher than one would expect. As there are many *normal* patterns in the 10% partition, this suggests that the patterns are not entirely separable from the patterns of the other classes. This is a reasonable assumption as the goal of any attack is to appear normal. The most common *normal* misclassifications are *neptune* (*DoS*), *portsweep* (*probe*) and *teardrop* (*probe*) (see table B.4).

The *DoS* frequency of selection is exactly what one would expect. It is made up of two very large classes and four smaller classes. Figure 4.16 is a reproduction of figure 4.15, except that the types are now divided into their individual classes. There, the bars correspond to (*normal*: *normal*); (*DoS*: *neptune*, *smurf*, *pod*, *teardrop*, *land*, *back*); (*probe*: *portsweep*, *ipsweep*, *satan*, *nmap*); (*U2R*: *bufferoverflow*, *loadmodule*, *perl*, *rootkit*); and (*R2L*: *guesspasswd*, *ftpwrite*, *imap*, *phf*, *multihop*, *warezmaster*, *warezclient*, *spy*), in that order. *Neptune* and *smurf* dominate *DoS*, comprising ninety-nine percent of all *DoS* patterns, and in fact almost eighty percent of the entire training partition. They are allocated plenty of resources by the SOM and are easily learned. The other four DoS classes have their selection frequencies on par with the other non-DoS attack classes. Nearly all of the DoS corrected partition predic-

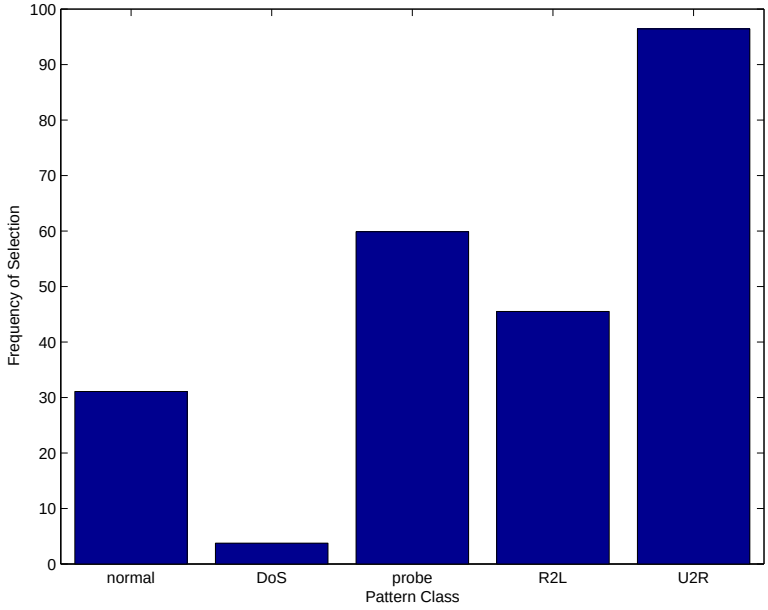


Figure 4.15: DSSSOM pattern selection frequency, by pattern type, for the KDD-99 41-feature data set.

tive error is caused by the unseen (in the 10% partition) *mailbomb*, *processtable*, and *apache2* classes, suggesting that these *DoS* attacks are quite different from the others. All three are most commonly misclassifed as *normal* (see figure B.5). It relation to training time, it is interesting to note that the average *smurf* or *neptune* pattern is seen only a few times during training, whereas it is seen 4000 times in Kayacik's SOM. This is a clear demonstration of the effect of DSS and the training time wasted by the SOM. The *U2R* frequency of selection is exactly what one might expect given that *U2R* patterns comprise only 0.01% of the 10% partition. All *U2R* patterns are misclassifed as *normal* (see table B.8), suggesting that the class type is simply too small to be learned by the DSSSOM (even with the added emphasis of DSS). It is also possible that the patterns are too similar to *normal* to be differentiated from *normal* by the SOM.

The *R2L* patterns are relatively easy compared to *probe* and *U2R*, given their respective selection frequencies. Still, the DSSSOM achieves very poor *R2L* predictive accuracies. One might assume that *R2L* is simply too similar to *normal*, but the good *R2L* 10% partition accuracy suggests that the *R2L* patterns in the 10% partition are not representative of the KDD-99 *R2L* patterns in general. This hypothesis is backed

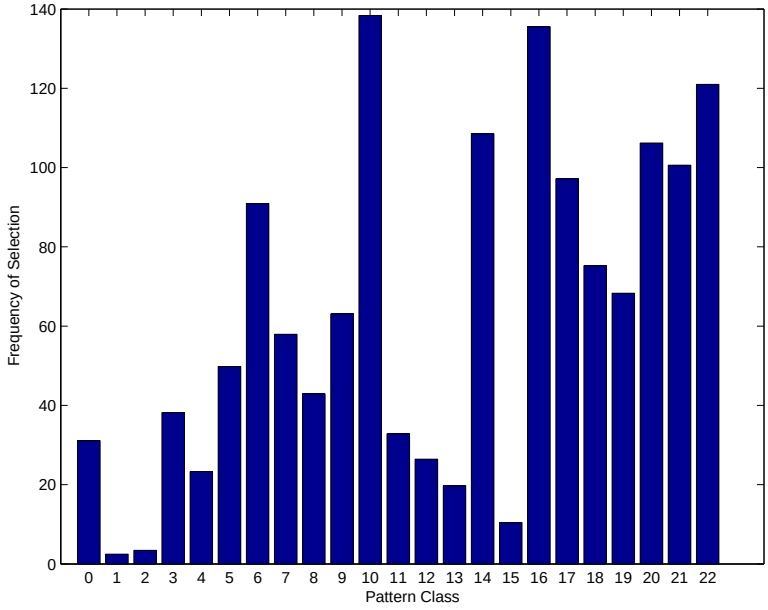


Figure 4.16: DSSSOM pattern selection frequency, by pattern class, for the KDD-99 41-feature data set.

up by the distribution of the pattern classes over the 10% and corrected partitions (see table B.7). There, it can be seen that over ninety percent of *R2L* patterns in the 10% partition are in classes that do not appear at all in the corrected partition. In fact, the 16347 R2L patterns appearing in the corrected partition belong to classes of which there are only 104 (of 1126) patterns in the 10% partition. Table B.7 shows that all of the 7741 patterns in the *snmpgetattack* class (which only appears in the corrected partition) are misclassified as *normal*. The same is true for the 2406 patterns in *snmpguess*. A similar situation is also true for *warezmaster* and *guesspassword* (which appear in the 10% partition with very low frequency, and in the corrected partition with much higher frequency). The only exception to the poor performance is the *httptunnel* class (which has no patterns in the 10% partition); two-thirds of the 158 patterns are classified as an attack (though none are classified as *R2L*).

Mediocre *probe* accuracies were achieved for both the 10% and corrected partitions. It appears that although the SOM is able to differentiate some of the *probe* patterns from *normal*, the remaining patterns are simply not separable. No good *probe* accuracies (higher than those reported) were achieved during experimentation. The *nmap*, *mscan* and *ipsweep* classes prevent such accuracies from being achieved.

Mscan is not seen in the 10% partition; the other two however are seen in decent numbers, yet they are not learned well. *Saint* also does not appear in the 10% partition, yet the *saint* accuracy is good. Thus, *saint* must be similar to another attack (likely *satan*, as shall be seen in the following subsection), while *nmap* is not. *Mscan* and *ipsweep* are not differentiated from *normal*. In table B.6, it can be seen that the vast majority of misclassification is indeed as *normal*.

The hit histograms for the first level of the first DSSSOM in table 4.16 can be seen figure 4.17. Figure 4.18 is a pattern label visualization that summarizes the hit histograms. The hit histograms show good separation over the classes. Most notably, notice that a large portion of *U2R* is very separable from *normal*. However, *normal* has small counts (relative to its class size, but large relative to the *U2R* class size) at many neurons, including the *U2R* neurons. Therefore, if there were more *U2R* patterns in the data set, a good *U2R* predictive accuracy could likely be achieved. The KDD-99 data set simply has too few *U2R* patterns even with the demonstrated added emphasis of DSS, and ultimately only one neuron is labeled *U2R*.

R2L does not achieve as clean a separation. However, it has already been demonstrated that the *R2L* accuracies suffer because of poor training data set representation. If this representation were better, higher testing accuracies would be achieved, as it appears that the *R2L* patterns are at least partially-separable from *normal*.

The separation between *normal* and *DoS* is good. Notice that the largest *DoS* count is away from the *normal*-dominated portion of the map, and that both class types have smaller counts in different areas on the right-hand side of the map (both form an "E" shape, and the shapes do not overlap).

Probe is not very separable from *normal*. It can be seen that the most populous regions of the *probe* hit histogram overlap with *normal*. However, the *probe* predictive accuracies are saved to an extent by the domination of *DoS* over *normal* for the two neurons having the highest *probe* counts (see figure 4.18). If all *DoS* patterns were removed from the data set, *probe* would not perform well at all. It is simply too similar to *normal*. In fact, early experimentation was performed on a KDD-99 10% subset consisting of only *normal* and *probe* for which the *probe* predictive accuracies were much worse.

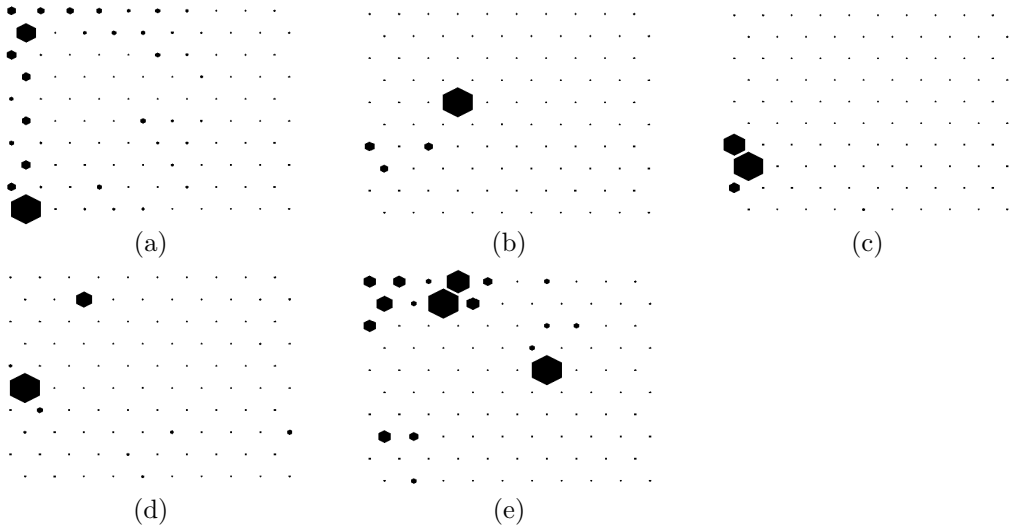


Figure 4.17: For a DSSSOM trained on the KDD-99 41-feature data set, hit histograms for the class types (a) *normal*, (b) *DoS*, (c) *probe*, (d) *R2L* and (e) *U2R*.

A constant observed over all KDD-99 41-feature experimentation was the inverse relationship that exists between *normal* and the *attack* class types. Building a second level in a DSSSOM hierarchy invariably meant a trade-off between the false positive and detection rates. Lower false positive rates and higher detection rates than those reported were achieved individually, but never in the same experiment.

4.3.5 The KDD-99 Six-Feature Data Set

Twenty DSSSOMs were trained for each of the six KDD-99 10% partition features. Twenty second-level DSSSOMs were then trained on top the 120 first-level DSSSOMs, with six distinct first-level DSSSOMs feeding each second-level DSSSOM. The first-level shift register size was set at 96, with taps at every fifth location, for a first-level input pattern dimension of 20. Third levels were built on top of the twenty second-level DSSSOMs, in the same fashion as were the second-level DSSSOMs for the 41-feature version of the data set. This was repeated for three second-level DSSSOM sizes, namely 10x10, 8x8 and 6x6. The first-level DSSSOMs were uniformly of size 6x6, and the third-level DSSSOMs were uniformly of size 10x10. The construction of a KDD-99 six-feature three-level hierarchy was described in section 4.1. The third-level results are presented here in the same fashion as were the 41-feature second-level

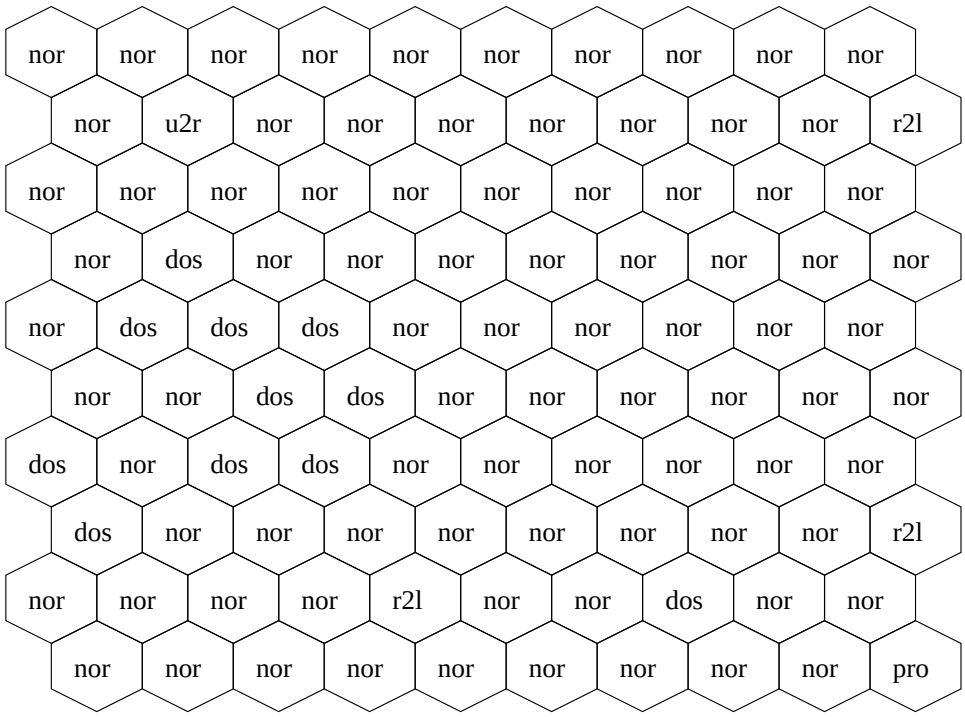


Figure 4.18: For a DSSSOM trained on the KDD-99 41-feature data set, the neuron label assignments.

results.

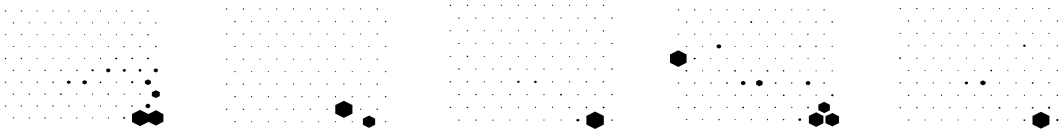
Tables 4.18 and 4.19 present the class type predictive accuracies for the best DSSSOM hierarchies of each size, on the corrected and 10% partitions, respectively. The first thing to note is that the performance on the six-feature data set is poor with respect to detection rate. However, it is unreasonable to expect good results given that a large portion of the training data set has been removed. The 6x6 corrected partition result is good compared to Kayacik's SOM result. The detection rate is 0.013 higher, but the false positive rate is 0.034 lower. It is important to have a very low false positive rate here because in intrusion detection, the end users of a detection system cannot be expected to respond to an overwhelming number of false alarms. Thus, it is wise to take a poorer detection rate in exchange for a better false positive rate. Having said that, the result achieved here is actually quite a bit better than Kayacik's result.

Table 4.18: DSSSOM three-level hierarchy training times and predictive accuracies on the KDD-99 six-feature corrected partition.

	Accuracy							*Time*
Size	*Normal*	*DoS*	*Probe*	*R2L*	*U2R*	*FP*	*Det.*	*(s)*
10x10	0.956	0.943	0.470	0.027	0.043	0.044	0.875	36303
8x8	0.992	0.933	0.409	0.006	0.000	0.008	0.863	30846
6x6	0.989	0.946	0.446	0.008	0.000	0.011	0.877	16340

Table 4.19: DSSSOM three-level hierarchy predictive accuracies on the KDD-99 six-feature 10% partition.

	Accuracy						
Size	*Normal*	*DoS*	*Probe*	*R2L*	*U2R*	*FP*	*Det.*
10x10	0.950	0.996	0.739	0.248	0.629	0.050	0.993
8x8	0.983	0.995	0.545	0.330	0.000	0.017	0.988
6x6	0.979	0.995	0.588	0.299	0.000	0.021	0.988



(a) (b) (c) (d) (e)

Figure 4.19: For a DSSSOM trained on the KDD-99 six-feature data set, hit histograms for the class types (a) *normal*, (b) *DoS*, (c) *probe*, (d) *R2L* and (e) *U2R*.

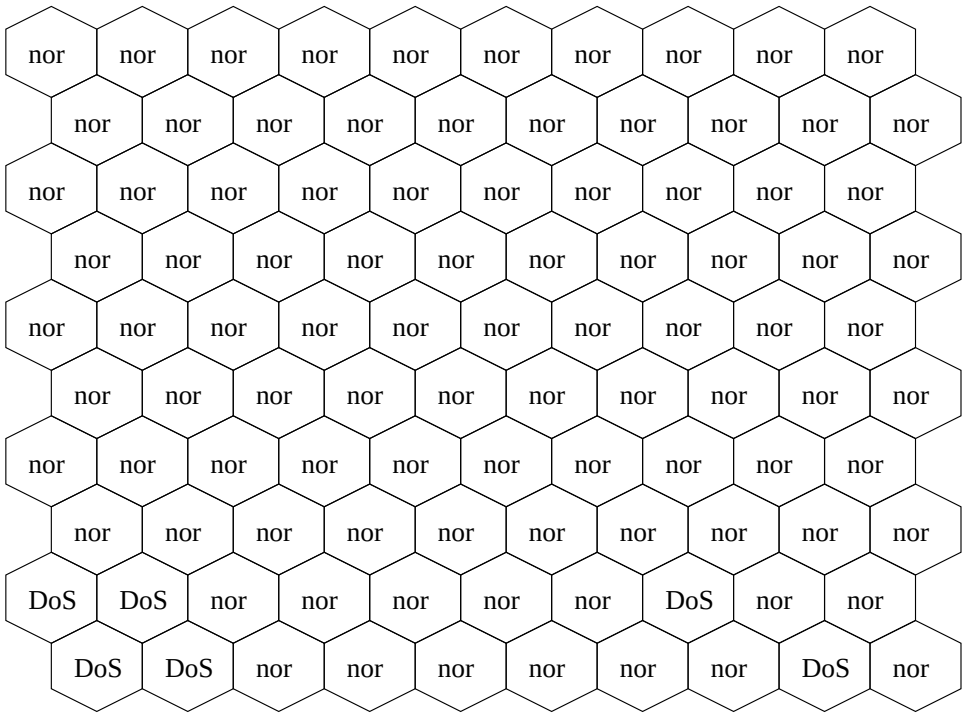


Figure 4.20: For a second-level DSSSOM trained on the KDD-99 six-feature data set, the neuron label assignments.

Hit histograms were generated for the second-level DSSSOM belonging to the three-level DSSSOM hierarchy from the first row of table 4.18. They are presented in figure 4.19. There, it can be seen that all five class types tend to have high counts toward the bottom-right of the map. Note that for the large *DoS* and *normal* classes, even the small hexagons represent relatively high counts. A pattern label visualization for the same DSSSOM is shown in figure 4.20. There, it can be seen that *normal* dominates nearly the entire map. The two neurons having the highest *DoS* counts are indeed labeled *DoS*, as are several to the bottom-left of the map, where the *normal* patterns are relatively sparse. The other three class types do not have high enough counts at any of the neurons to warrant labels. Fortunately, the third level of the hierarchy achieves more separation over the class types, allowing for very limited detection of the three types. An example of this separation can be seen in figure 4.21. There, the third-level DSSSOM built on top of the 97^{th} neuron (the third neuron from the right in the last row of figure 4.20) can be seen. The second-level neuron is labeled *normal*, however, at the third level, the map is successfully split into *normal*, *DoS*, and *probe* sections. This is the case for most of the third-level maps, which allows modest *probe* and better *DoS* predictive accuracies to be achieved.

Figure 4.22 presents the selection frequencies of the class types for DSSSOM from which the hit histograms and pattern label visualization were generated. The order of the classes in the figure is the same as it was in figure 4.16. Figure 4.23 summarizes the individual frequencies by class type. It can be seen that the relative frequencies of all five class types are identical to those of the 41-feature data set in figure 4.15. This suggests that the latter 35 features have little, if any, effect on the relative difficulties of the class types.

The individual class accuracies on the 10% and corrected six-feature partitions are available in tables B.10-B.15. In table B.10, it can be seen that *normal* is most often misclassified as *neptune*. This makes sense given the size of the *neptune* class. However, *smurf* is even larger and has no *normal* misclassifications. Thus, it can be said that *normal* is more similar to *neptune* than it is to *smurf*.

The *DoS* accuracies can be seen in table B.11. The accuracies are lower than those achieved with the 41-feature data set. Again, the *smurf* and *neptune* accuracies are

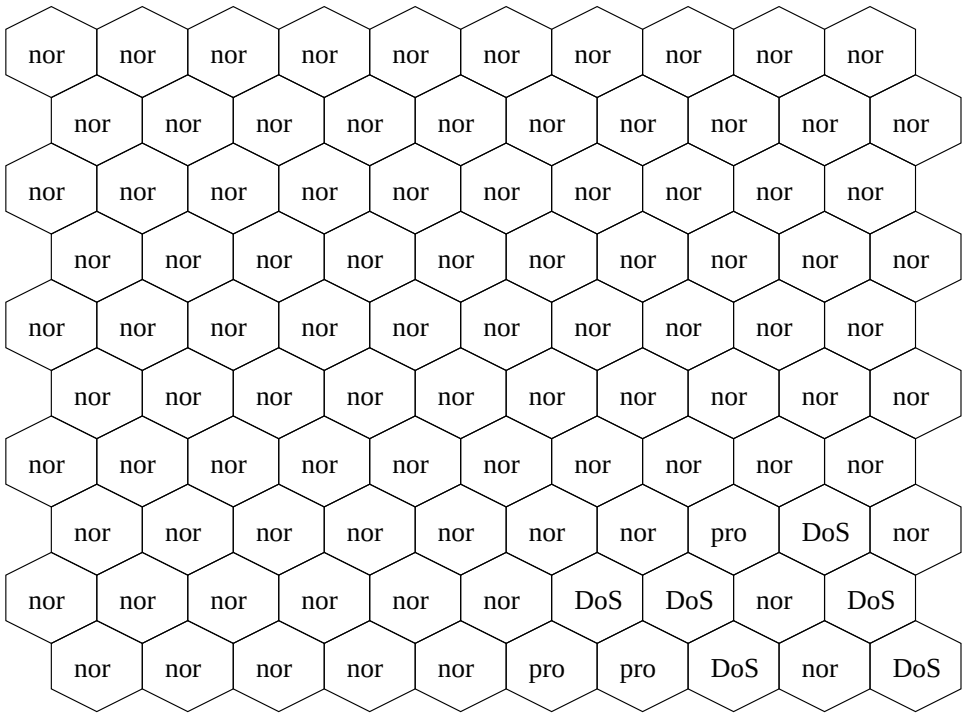


Figure 4.21: For a third-level DSSSOM trained on the KDD-99 six-feature data set, the neuron label assignments.

the highest, while the unseen classes (those with no patterns in the 10% partition) are the lowest. However, whereas the DSSSOM was able to achieve good *back* and *pod* accuracies with the 41-feature data set, it was not with the six-feature data set. Almost all *DoS* misclassification was as *normal*, which is hardly surprising given that the other three class types have very few labeled neurons over the entire three-level hierarchy.

Table B.12 presents the six-feature probe accuracies. There it can be seen that again the six-feature accuracies are lower than the 41-feature accuracies. *Saint* and *satan* again performed well, but the other classes did not. Of note is *portsweep*, on which a good 41-feature accuracy was achieved, but on which a poor six-feature accuracy was achieved. The bulk of the *probe* misclassifications were as *normal*, though a significant number were as *neptune*. *Saint* and *satan* were particularly susceptible to being classified as *neptune*. The two classes are (though their names would not suggest) undoubtedly similar to one another in both the input and output spaces, and to a lesser extent to *neptune*.

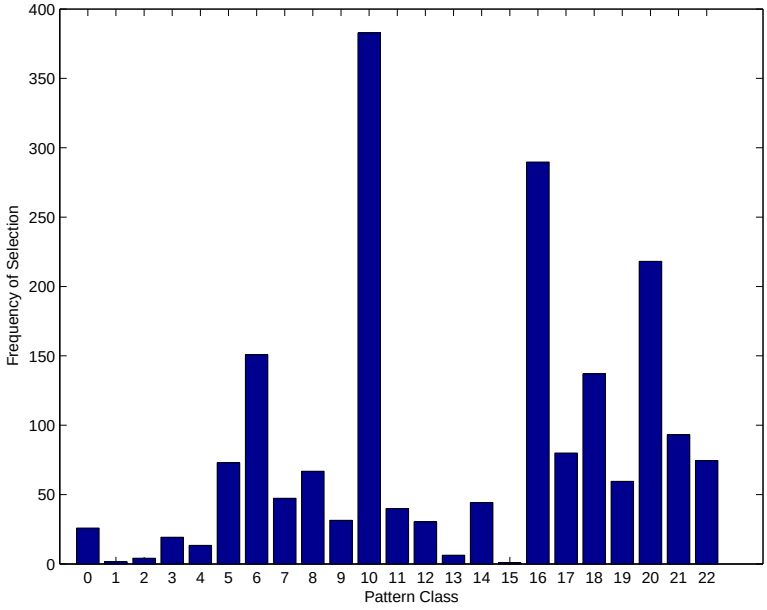


Figure 4.22: DSSSOM pattern selection frequency, by pattern class, for the KDD-99 41-feature data set.

The only *R2L* class for which a good 41-feature predictive accuracy is achieved is the unseen *httptunnel*. Unfortunately, none of the *R2L* classes achieve good six-feature accuracies, as table B.13 shows. Nearly all misclassifications are as *normal*, with several *neptune* and *ipsweep* misclassifications. The same is nearly true for U2R in table B.14, where all accuracies are zero, and all misclassifications are as *normal* (this is exactly what happened for the 41-feature data set).

Finally, though no SOM experimentation was performed on the six-feature data set, it is still possible to compare training times. Each SOM in Kayacik's three-level hierarchy was trained for 4000 epochs. Given a data set of size approximately 494021 (KDD-99 10% partition), the total number of patterns seen at the first level is just below two billion per SOM, or about twelve billion in total. The first level of the DSSSOM hierarchy in the third row of table 4.18 saw a total of just over 70 million patterns, or a factor of 168.5 fewer patterns. Training at the first level is therefore much faster for the DSSSOM, allowing for the added expense of DSS computation (which, through time experimentation on the DSSSOM implementation, was seen to add less than one percent to the overall program computation). At the second level, a constructed SOM trained for 4000 epochs sees about two billion first-level

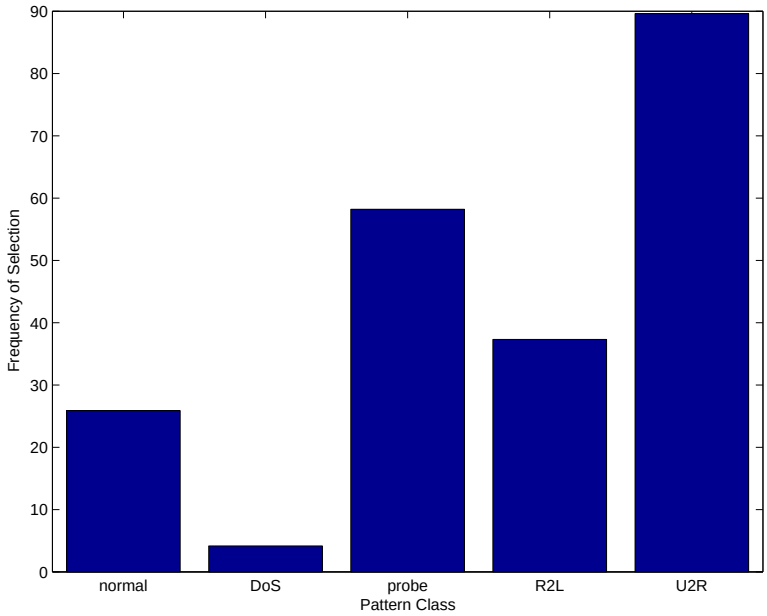


Figure 4.23: DSSSOM pattern selection frequency, by pattern class type, for the KDD-99 41-feature data set.

outputted patterns. The DSSSOM hierarchy saw 4.08 million such patterns, a factor of about 484 fewer. There is relatively little computation involved in constructing the third-level, and as the number of patterns learned by the SOM at the third level is unknown, speed-up estimates at that level are not possible. However, it is already plain to see that the DSSSOM is much faster than the SOM. In fact, the three-level DSSSOM trains in just over 4.5 hours, which is a factor of 7.3 faster than the two-level 41-feature SOM reported in the previous section. Though the two- and three-level hierarchies cannot be compared directly, this is a good indication of just how much training is sped up by the addition of DSS. Consider finally one further statistic: in order for the SOM training times to be brought on par with the DSSSOM training times, the first level of an SOM hierarchy would need to be trained for no more than 24 epochs, and the second level for no more than eight epochs. A good representation of the KDD-99 data set cannot be achieved in such a short amount of time by the SOM.

5 Conclusions and Future Work

The DSSSOM is the marriage of DSS and the SOM, facilitated by modifications to both. DSS is a method of focusing a GP on the particularly difficult-to-learn patterns in a given data set, without ignoring any non-difficult patterns completely. Two pattern measures are key: difficulty and age. The DSS algorithm is iterative, where in each iteration a subset of the data set is formed consisting of the most difficult and old patterns. The subset is fed to the GP, and the difficulties and ages of the applicable patterns are updated. Pattern difficulty is a GP-specific measure and is updated only when selection to a subset has taken place; pattern age is incremented each time a pattern is not selected to the current subset, and reset to one when the pattern is selected.

In applying DSS to the SOM, pattern difficulty is redefined to be the Euclidean distance from a pattern to its BMU. In order for this to be meaningful early in SOM training, a brief period of initialization is needed during which the SOM is trained in the traditional fashion. Pattern age remains unchanged. As BMU distances are retrieved during training and used to influence the selection of the next subset, DSS cannot be simply laid on top of the SOM algorithm. A traditional SOM was implemented into which DSS was integrated, and with which all original SOM and DSSSOM experimentation took place.

DSS made possible the use of a new SOM training stopping condition. As the average difficulty of the patterns in the data set can be expected to decrease as the SOM prototype vectors are adjusted to better represent them, the ordering phase of training can be stopped when a linear mapping from the average difficulty to the learning rate and neighbourhood radius yields a learning rate below a specified threshold. To ensure that the threshold is achieved, average difficulty leveling out is detected and the learning rate and radius forced lower in small incremental steps. After the threshold is reached, fine-tuning takes place for twice as many epochs as were spent on ordering. This mechanism inherently provides a way of computing the learning rate and radius, for the computation of which a specified training length is traditionally used.

As DSS requires access to the entire training data set at all times, very large data sets that cannot fit into memory cannot be learned. To solve this problem, blocks of such a data set are read in one at a time. DSS is then performed only on the current block, for a specified period of time, after which a new block is read in. Whereas the selection of blocks is traditionally random, here DSS is also used to select blocks. The difficulty of a block is simply the average difficulty of the patterns within it; age is defined as before.

Preliminary experiments were conducted that demonstrated the higher frequency of selection of less common and more difficult patterns and blocks for the KDD-99 41-feature data set. Having verified the correctness of the DSSSOM, the purpose of further experimentation was to show that it performs as well as the traditional SOM while requiring less training time. Experiments were performed on five data sets: adult, covertype, shuttle, KDD-99 41-feature, and KDD-99 six-feature. Performance was measured as overall predictive accuracy for the previous three data sets, and as a combination of the false positive and detection rates for the latter two.

The adult and covertype data sets served to demonstrate two things: that DSS cannot help the SOM to represent data that it is inherently unable to represent, and that the two algorithms achieve very similar predictive accuracies. On the adult data set, the SOM achieved a predictive accuracy of 0.774, while the DSSSOM achieved 0.776. On the covertype-a data set, the respective values were 0.306 and 0.274. DSSSOM hierarchies were unable to attain accuracies on par with other learning algorithms in either case, though only one other clustering result is available between them for comparison. DSS determined that the two classes in the adult data set have the same difficulty, largely due to the fact that high counts of both classes fell on most neurons. There was some variation over class difficulty for covertype-a; oddly, the largest class was found to be the most difficult, and a good accuracy for that class (and therefore the entire data set) was not achieved. Even though DSS focused the SOM on the difficult patterns and classes, the SOM could not achieve good separation over the classes.

The shuttle data set also served to demonstrate that the DSSSOM performs as well as the SOM. There, the SOM achieved an overall accuracy of 0.963, while the

DSSSOM achieved 0.968. The best DSSSOMs had training times far below those of the best SOMs. SOMs trained for much shorter times, however, enjoyed accuracies that were not far below 0.963. Still, the DSSSOM training times were impressive in that they were lower than SOMs trained for only fifty epochs. The best overall accuracy achieved by a DSSSOM hierarchy was 0.995, which is within 0.001 of the best existing clustering result.

Unlike the previous three data sets, the KDD-99 41-feature data set serves to demonstrate all of the benefits of using the DSSSOM. First, previous work has demonstrated that it is learnable by the SOM. Second, it has been demonstrated that it is not a simple data set to learn. No algorithm has ever achieved a detection rate higher than 0.920 nor a false positive rate lower than 0.005. Third, the data set is large, and as such it can be used to demonstrate DSSSOM block and pattern selection characteristics. Kayacik's SOM hierarchy achieved false positive and detection rates of 0.906 and 0.016; a DSSSOM hierarchy respectively achieved 0.904 and 0.005. For time comparison purposes, a single SOM hierarchy was trained using the same parameters as were used by Kayacik. The DSSSOM was found to require a factor of ten less training time than the SOM. The DSSSOM is shown to select the smaller and more difficult pattern classes (e.g., *nmap*) with much greater frequency than the largest and easiest classes (e.g., *smurf* and *neptune*). It was shown that the most common misclassification of attack classes is as *normal*. The *R2L* patterns in the 10% data set were shown to not be representative of all *R2L* patterns in KDD-99 41-feature. The *U2R* classes were found to be too small to learn, even with the demonstrated added impact of DSS. Most *probe* classes were either poorly represented in the 10% partition, or too similar to *normal* to yield good accuracies. *Smurf*, *neptune*, *back* and *pod* account for all of *DoS*' excellent performance. The other *DoS* classes were underrepresented in either the 10% or corrected partition.

The KDD-99 six-feature data set is a pruned version of the 41-feature data set. Kayacik achieved detection and false positive rates of 0.890 and 0.046, respectively. The DSSSOM respectively achieved rates of 0.877 and 0.011. For both data sets, the DSSSOM had a much better false positive rate, and in both cases the detection rate was slightly worse. The DSSSOM was found to require a factor of over 150 times less

patterns seen during first-level training, and a factor of over 450 less at the second-level. The class type selection frequencies were observed to be the same as those of the DSSSOM on the 41-feature data set. In general, the individual class accuracies were similar to those achieved for the 41-feature data set, though good accuracies on the six-feature data set were not achieved for the *back*, *pod*, *portsweep* and *httptunnel* classes, whereas they were for the 41-feature version. The relative performances of the other classes and the reasoning behind the individual class (and type) performances were found to be the same for the six-feature data set.

Despite the immense promise of DSS applied to the SOM, there is still a great deal of experimentation that needs to be conducted to determine exactly how and when it should be used. How a DSSSOM is used is determined by its parameter set. When it is used is determined by the data set to be learned. The parameter set of the DSSSOM was kept very narrow in this work. In particular, the probabilities with which patterns and blocks were selected by difficulty were kept constant. Better predictive accuracies might be achieved for the shuttle and KDD-99 data sets if alternate probabilities had been tried. For example, even with a 0.9 probability of pattern selection by difficulty, the KDD-99 SOMs saw *far* more *smurf*, *neptune* and *normal* patterns than the patterns of the other classes, due to their extremely large sizes. DSSSOM map sizes were kept within a small range to ensure feasible traditional SOM training times and valid comparison with previous work. However, the fast DSSSOM algorithm affords the use of larger map sizes, especially at the higher levels of the DSSSOM hierarchies.

The DSSSOM has been shown to be most suited to large, non-trivial data sets, such as the KDD-99 data sets. It would be of great use to demonstrate the ability of the DSSSOM on other such data sets. It is still uncertain as to whether the DSSSOM can yield significantly better performance (e.g., predictive accuracy) than the SOM. None of the data sets on which experiments were performed were truly ideal for DSSSOM experimentation. Ideally, an unbalanced data set is needed that is neither simple nor impossible for the SOM algorithm to learn, yet has good representation in its training and test partitions. It should not be difficult to find such a data set, but unfortunately none are known to be publicly available at this time. Clearly, such data sets will be

plentiful in the future, as more data sets become available. Experimentation on such data sets would prove invaluable.

It would be interesting to demonstrate the effectiveness of the DSSSOM as a data reduction tool. After the completion of DSSSOM training, the patterns of the training data set are all associated with particular difficulties. These difficulties can be used to generate a smaller data set. Ideally, all patterns beyond a certain difficulty are selected to the new data set, as well as a small subset of the easy patterns. The easy patterns must be included to ensure good representation of the entire data set. Alternatively, frequency of selection to a DSS subset can be used in place of pattern difficulty. Either of these effectively act as measures of a pattern's importance in forming a good representation of the particular data set. Importance is of course biased toward the SOM, but the generated data sets will lead to faster training times and perhaps better results for other learning algorithms. It might be worthwhile to train another DSSSOM on the generated data set to determine whether the results improve given a more concise data set.

Future work should also focus on the DSSSOM stopping mechanism. While the mechanism presented in this work is shown to work well, it might stand to benefit from some tweaking. There are three avenues down which to proceed. The first involves determining the ideal number of epochs over which the average block difficulty is examined in determining whether it has leveled out. In this work, this number was arbitrarily set at 100 epochs. Reducing this number will result in lower DSSSOM training times for difficult data sets. Increasing the number will slow training times but perhaps lead to better predictive accuracies.

The second avenue involves determining the ideal minimum training time. In this work, the minimum number of DSSSOM epochs was set arbitrarily at 100. Decreasing this number will speed up training without reducing the predictive accuracy for very small and very easy data sets. Increasing this number will result in more realistic pattern and block difficulty values early during DSSSOM training, and further ensure that all patterns in the training data set are seen. It might be a good idea to base the minimum number of epochs on the size of the data set in use. This is particularly applicable to training the upper level of a DSSSOM hierarchy, where the data sets

may vary immensely in size.

The third avenue involves changing the way in which the average block difficulty is computed. In order to ensure that all classes are learned, it might be a good idea to weight the contribution of each class to the average block difficulty. As an example, suppose the data set in use is dominated by a single class. Suppose then that the large class is easy to learn. Then, during training, the average block difficulty will plummet, and training will quickly stop. Even if the average difficulties of the patterns in the other classes are high, their contribution to the overall average difficulty may be too small to stop training. Thus, it may be desirable to compute the average difficulty as a weighted sum of the average class difficulties. Then, even after a particular class has been learned well, training will continue to learn the remaining classes.

A promising avenue of future work would be to investigate the combination of the DSSSOM with another SOM speed-up technique. For example, a clustering-based initialization mechanism would serve the DSSSOM well. Pattern difficulty has very little meaning early during training, because the DSSSOM has not yet achieved a good ordering of its prototype vectors. Traditional SOM initialization is currently used to try to obtain a crude ordering. However, clustering-based initialization can achieve a better ordering in a shorter amount of time. After the DSSSOM has achieved a good ordering (and the ordering phase of training has completed), a shortcut winner search such as the one described by Kohonen in section 2.6, can be employed to speed up fine-tuning. In practice, fine-tuning generally consumes two-thirds of the total training time. Thus, a speed-up in fine-tuning would significantly speed-up training as a whole.

In this work, the application of DSS to the SOM has been investigated. DSS has not previously been applied to any learning algorithms other than GP. The exploration of its application to other algorithms should be explored; if it can speed up those algorithms as well, the exploration would be very worth-while. The only requirement for any algorithm is a method of computing a pattern's difficulty.

The DSSSOM is a fast learning algorithm based on the SOM. It has been shown to achieve predictive accuracies on par with the SOM with less training time. The accuracies are slightly better than those of the SOM for data sets that are well-

learned by both algorithms. The DSSSOM, given a large and not too difficult data set, is shown to achieve training times far below those associated with the SOM. The DSSSOM is not limited by data set size through the use of DSS-powered block selection. It also does not require a specified training length, as it uses the DSS concept of difficulty to stop training at the proper time. It is highly configurable, with numerous parameters that can be tweaked to achieve the desired result in any learning situation. The DSSSOM it is an extremely promising technology that has the potential to replace the traditional SOM in most of its applications.

Appendix A

Table A.1: Adult data set compositions.

	Number of Patterns	
Pattern Label	*Training*	*Test*
$\leq 50k$	22654	11360
$> 50k$	7508	3700
TOTAL	30162	15060

Table A.2: Covertype data set compositions.

	Number of Patterns		
Pattern Label	*Training*	*Validation*	*Test*
Spruce/Fir	1620	540	209680
Lodgepole Pine	1620	540	281141
Ponderosa Pine	1620	540	33594
Cottonwood/Willow	1620	540	587
Aspen	1620	540	7333
Douglas Fir	1620	540	15207
Krummholz	1620	540	18350
TOTAL	11340	3780	565892

Table A.3: Shuttle data set compositions.

	Number of Patterns	
Pattern Label	*Training*	*Testing*
Rad Flow	34108	11478
Fpv Close	37	13
Fpv Open	132	39
High	6748	2155
Bypass	2458	809
Bpv Close	6	4
Bpv Open	11	2
TOTAL	43500	14500

Table A.4: KDD-99 data set normal composition.

Number of Patterns	
10%	*Corrected*
97278	60593

Table A.5: KDD-99 data set DoS composition.

Pattern Label	*Number of Patterns*	
	10%	*Corrected*
apache2	0	794
back	2203	1098
land	21	9
mailbomb	0	5000
neptune	107201	58001
pod	264	87
processtable	0	759
smurf	280790	164091
teardrop	979	12
udpstorm	0	2
total	391458	229853

Table A.6: KDD-99 data set probe composition.

Pattern Label	*Number of Patterns*	
	10%	*Corrected*
ipsweep	1247	306
mscan	0	1053
nmap	231	84
portsweep	1040	354
saint	0	736
satan	1589	1633
total	4107	4166

Table A.7: KDD-99 data set R2L composition.

Pattern Label	*Number of Patterns*	
	10%	*Corrected*
ftpwrite	8	3
guesspassword	53	4367
httptunnel	0	158
imap	12	1
multihop	7	18
named	0	17
phf	4	2
sendmail	0	17
snmpgetattack	0	7741
snmpguess	0	2406
spy	2	0
warezclient	1020	0
warezmaster	20	1602
worm	0	2
xlock	0	9
xsnoop	0	4
total	1126	16347

Table A.8: KDD-99 data set U2R composition.

Pattern Label	*Number of Patterns*	
	10%	*Corrected*
bufferoverflow	30	22
loadmodule	9	2
perl	3	2
ps	0	16
rootkit	10	13
sqlattack	0	2
xterm	0	13
total	52	70

Table A.9: KDD-99 data set overall composition.

Category	*Number of Patterns*	
	10%	*Corrected*
normal	97278	60593
DoS	391458	229853
probe	4107	4166
R2L	1126	16347
U2R	52	70
total attack	396743	250436
grand total	494021	311029

Appendix B

Table B.1: SOM training times and predictive accuracies for the adult testing partition.

Size	*Training Length (epochs)*	*Accuracy*			*Time (s)*
		$\leq$ \$50k	$>$ \$50k	*Overall*	
10x10	4000	0.993	0.095	0.772	10136
8x8	4000	0.995	0.083	0.771	6890
6x6	4000	0.997	0.040	0.762	4350
10x10	1000	0.997	0.089	0.774	2509
8x8	1000	0.995	0.085	0.771	1721
6x6	1000	0.996	0.025	0.758	1107
10x10	250	0.996	0.080	0.771	622
8x8	250	0.996	0.076	0.770	428
6x6	250	0.995	0.044	0.761	275
10x10	50	0.996	0.087	0.773	125
8x8	50	0.995	0.079	0.770	86
6x6	50	0.999	0.033	0.762	55
10x10	10	0.989	0.099	0.770	25
8x8	10	0.993	0.084	0.770	17
6x6	10	0.999	0.025	0.760	11
10x10	1	0.994	0.031	0.757	2
8x8	1	0.999	0.019	0.758	2
6x6	1	1.000	0.005	0.755	1

Table B.2: SOM training times and predictive accuracies for the covertype-a testing partition.

Size	*Tr.Len. (epochs)*	*Accuracy*								*Time (s)*
		S/F	*LPine*	*PPine*	*C/W*	*Asp*	*DFir*	*Krum*	*Overall*	
10x10	4000	0.370	0.187	0.288	0.652	0.529	0.459	0.615	0.287	9067
8x8	4000	0.388	0.191	0.422	0.668	0.450	0.197	0.552	0.293	6185
6x6	4000	0.281	0.081	0.305	0.719	0.436	0.260	0.537	0.193	4006
10x10	1000	0.417	0.163	0.288	0.656	0.529	0.459	0.589	0.292	2292
8x8	1000	0.401	0.164	0.408	0.687	0.535	0.185	0.475	0.282	1567
6x6	1000	0.274	0.095	0.217	0.796	0.431	0.228	0.539	0.191	1004
10x10	250	0.384	0.183	0.366	0.670	0.610	0.337	0.644	0.294	555
8x8	250	0.382	0.107	0.396	0.676	0.472	0.224	0.596	0.250	370
6x6	250	0.280	0.081	0.306	0.719	0.396	0.295	0.536	0.194	233
10x10	50	0.407	0.206	0.334	0.797	0.413	0.219	0.619	0.305	109
8x8	50	0.396	0.206	0.414	0.671	0.446	0.193	0.538	0.303	76
6x6	50	0.281	0.081	0.305	0.719	0.436	0.260	0.537	0.193	46
10x10	10	0.371	0.231	0.241	0.709	0.526	0.370	0.696	0.306	24
8x8	10	0.378	0.073	0.346	0.683	0.515	0.316	0.618	0.233	14
6x6	10	0.246	0.070	0.412	0.514	0.430	0.274	0.583	0.183	10
10x10	1	0.006	0.000	0.008	0.416	0.011	0.234	0.979	0.041	2
8x8	1	0.001	0.000	0.007	0.450	0.012	0.247	0.981	0.040	1
6x6	1	0.000	0.001	0.037	0.394	0.010	0.284	0.963	0.042	1

Table B.3: SOM training times and predictive accuracies for the shuttle testing partition.

Size	*Tr.Len. (epochs)*	*Accuracy*								*Time (s)*
		RFlow	*FClose*	*FOpen*	*High*	*Byp*	*BClose*	*BOpen*	*Ovr.*	
10x10	4000	0.965	0.154	0.026	0.896	0.989	0.000	0.000	0.952	11281
8x8	4000	0.981	0.308	0.000	0.762	0.994	0.000	0.000	0.946	7525
6x6	4000	0.951	0.000	0.000	0.771	0.989	0.000	0.000	0.923	4610
10x10	1000	0.981	0.154	0.231	0.871	0.994	0.000	0.000	0.963	2826
8x8	1000	0.956	0.000	0.000	0.878	0.991	0.000	0.000	0.943	1883
6x6	1000	0.951	0.000	0.000	0.771	0.989	0.000	0.000	0.923	1150
10x10	250	0.988	0.000	0.128	0.807	0.993	0.000	0.000	0.957	706
8x8	250	0.962	0.231	0.487	0.797	0.975	0.000	0.000	0.936	474
6x6	250	0.924	0.000	0.000	0.667	0.995	0.000	0.000	0.886	288
10x10	50	0.975	0.154	0.256	0.877	0.991	0.000	0.000	0.958	141
8x8	50	0.976	0.000	0.026	0.783	0.979	0.000	0.000	0.943	94
6x6	50	0.943	0.000	0.000	0.698	0.995	0.000	0.000	0.906	58
10x10	10	0.970	0.077	0.282	0.879	0.994	0.000	0.000	0.955	28
8x8	10	0.955	0.231	0.000	0.781	0.892	0.000	0.000	0.922	18
6x6	10	0.947	0.000	0.000	0.669	0.995	0.000	0.000	0.905	11
10x10	1	0.978	0.000	0.000	0.395	0.269	0.000	0.000	0.848	3
8x8	1	0.999	0.000	0.000	0.747	0.000	0.000	0.000	0.902	2
6x6	1	0.995	0.000	0.000	0.820	0.000	0.000	0.000	0.910	1

Table B.4: DSSSOM hierarchy normal predictive accuracies and three most common misclassifications on the KDD-99 41-feature corrected partition.

Class	*10%*		*Corrected*				
	Count	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
normal	97278	0.999	60593	0.995	neptune(116)	portsweep(109)	teardrop(34)

Table B.5: DSSSOM hierarchy DoS predictive accuracies and three most common misclassifications on the KDD-99 41-feature corrected partition.

Class	*10%*		*Corrected*				
	Count	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
apache2	0	-	794	0.349	normal(517)	back(277)	-
back	2203	0.989	1098	0.973	normal(30)	-	-
land	21	0.000	9	0.222	normal(7)	portsweep(2)	-
mailbomb	0	-	5000	0.000	normal(5000)	-	-
neptune	107201	0.999	58001	0.998	teardrop(133)	normal(127)	portsweep(91)
pod	264	0.981	87	0.931	normal(6)	-	-
processtable	0	-	759	0.020	normal(744)	neptune(15)	-
smurf	280790	1.000	164091	1.000	normal(77)	-	-
teardrop	979	0.854	12	0.250	normal(9)	neptune(1)	-
udpstorm	0	-	2	0.000	normal(2)	-	-

Table B.6: DSSSOM hierarchy probe predictive accuracies and three most common misclassifications on the KDD-99 41-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
ipsweep	1247	0.490	306	0.258	normal(227)	smurf(3)	-
mscan	0	-	1053	0.256	normal(783)	neptune(151)	portsweep(71)
nmap	231	0.403	84	0.024	normal(82)	teardrop(2)	-
portsweep	1040	0.915	354	0.924	normal(27)	teardrop(4)	neptune(1)
saint	0	-	736	0.856	satan(570)	normal(106)	neptune(26)
satan	1589	0.984	1633	0.994	neptune(297)	portsweep(47)	normal(10)

Table B.7: DSSSOM hierarchy R2L predictive accuracies and three most common misclassifications on the KDD-99 41-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
ftpwrite	8	0.000	3	0.000	normal(3)	-	-
guesspassword	53	0.962	4367	0.005	normal(4344)	warezmaster(16)	rootkit(1)
httptunnel	0	-	158	0.671	portsweep(77)	normal(52)	neptune(29)
imap	12	0.000	1	0.000	normal(1)	-	-
multihop	7	0.000	18	0.000	normal(18)	-	-
named	0	-	17	0.000	normal(17)	-	-
phf	4	0.000	2	0.000	normal(2)	-	-
sendmail	0	-	17	0.000	normal(17)	-	-
snmpgetattack	0	-	7741	0.000	normal(7741)	-	-
snmpguess	0	-	2406	0.000	normal(2406)	-	-
spy	2	0.000	0	-	-	-	-
warezclient	1020	0.912	0	-	-	-	-
warezmaster	20	0.800	1602	0.004	normal(1596)	neptune(4)	teardrop(2)
worm	0	-	2	0.000	normal(2)	-	-
xlock	0	-	9	0.111	normal(8)	ipsweep(1)	-
xsnoop	0	-	4	0.000	normal(4)	-	-

Table B.8: DSSSOM hierarchy U2R predictive accuracies and three most common misclassifications on the KDD-99 41-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
bufferoverflow	30	0.300	22	0.000	normal(22)	-	-
loadmodule	9	0.000	2	0.000	normal(2)	-	-
perl	3	0.000	2	0.000	normal(2)	-	-
ps	0	-	16	0.000	normal(16)	-	-
rootkit	10	0.100	13	0.000	normal(13)	-	-
sqlattack	0	-	2	0.000	normal(2)	-	-
xterm	0	-	13	0.000	normal(13)	-	-

Table B.9: DSSSOM hierarchy confusion matrix for the KDD-99 41-feature corrected partition.

	Normal	*DoS*	*Probe*	*U2R*	*R2L*	*Overall*
Normal	60285	174	115	5	14	60593
DoS	6519	223166	168	0	0	229853
Probe	1235	527	2395	0	9	4166
R2L	16211	35	78	1	22	16347
U2R	70	0	0	0	0	70
Overall	84320	223902	2756	6	45	311029

Table B.10: DSSSOM hierarchy normal predictive accuracies and three most common misclassifications on the KDD-99 six-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
normal	85457	0.950	60521	0.956	neptune(1688)	ipsweep(192)	back (162)

Table B.11: DSSSOM hierarchy DoS predictive accuracies and three most common misclassifications on the KDD-99 six-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
apache2	0	-	794	0.186	normal(646)	neptune(109)	back(37)
back	1964	0.974	1098	0.586	normal(455)	ipsweep(1)	-
land	16	0.500	9	0.000	normal(9)	-	-
mailbomb	0	-	5000	0.002	normal(4988)	ipsweep(12)	-
neptune	106930	0.996	58001	0.911	normal(5151)	ipsweep(6)	satan(6)
pod	239	0.096	87	0.000	normal(87)	-	-
processtable	0	-	759	0.042	normal(727)	warezclient(15)	neptune(14)
smurf	280645	0.999	164091	0.993	normal(1177)	satan(71)	portsweep(3)
teardrop	763	0.419	12	0.000	normal(12)	-	-
udpstorm	0	-	2	0.000	normal(2)	-	-

Table B.12: DSSSOM hierarchy probe predictive accuracies and three most common misclassifications on the KDD-99 six-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
ipsweep	916	0.612	306	0.085	normal(280)	neptune(12)	smurf(10)
mscan	0	-	1053	0.100	normal(948)	neptune(64)	smurf(16)
nmap	124	0.492	84	0.012	normal(83)	ipsweep(1)	-
portsweep	816	0.501	354	0.037	normal(341)	neptune(9)	ipsweep(3)
saint	0	-	736	0.686	neptune(501)	normal(231)	phf(2)
satan	1463	0.969	1633	0.784	neptune(1280)	normal(353)	-

Table B.13: DSSSOM hierarchy R2L predictive accuracies and three most common misclassifications on the KDD-99 six-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
ftpwrite	6	0.500	3	0.000	normal(3)	-	-
guesspassword	43	0.023	4367	0.010	normal(4325)	ipsweep(28)	neptune(7)
httptunnel	0	-	158	0.114	normal(140)	neptune(14)	ipsweep(3)
imap	12	0.000	1	0.000	normal(1)	-	-
multihop	4	0.750	18	0.056	normal(17)	smurf(1)	-
named	0	-	17	0.000	normal(17)	-	-
phf	4	0.250	2	0.000	normal(2)	-	-
sendmail	0	-	17	0.000	normal(17)	-	-
snmpgetattack	0	-	7718	0.024	normal(7529)	neptune(63)	ipsweep(63)
snmpguess	0	-	2406	0.014	normal(2373)	smurf(15)	neptune(9)
spy	2	0.000	0	-	-	-	-
warezclient	478	0.259	0	-	-	-	-
warezmaster	8	0.750	1602	0.025	normal(1562)	neptune(20)	ipsweep(16)
worm	0	-	2	0.000	normal(2)	-	-
xlock	0	-	9	0.000	normal(9)	-	-
xsnoop	0	-	4	0.000	normal(4)	-	-

Table B.14: DSSSOM hierarchy U2R predictive accuracies and three most common misclassifications on the KDD-99 six-feature corrected partition.

	10%		*Corrected*				
Class	*Count*	*Acc.*	*Count*	*Acc.*	*Top Misclassifications*		
bufferoverflow	20	0.700	22	0.000	normal(22)	-	-
loadmodule	3	0.000	2	0.000	normal(2)	-	-
perl	3	0.333	2	0.000	normal(2)	-	-
ps	0	-	16	0.000	normal(16)	-	-
rootkit	9	0.778	13	0.000	normal(13)	-	-
sqlattack	0	-	2	0.000	normal(2)	-	-
xterm	0	-	13	0.000	normal(13)	-	-

Table B.15: DSSSOM hierarchy confusion matrix for the KDD-99 six-feature corrected partition.

	Normal	*DoS*	*Probe*	*U2R*	*R2L*	*Overall*
Normal	57845	2133	447	96	0	60521
DoS	13197	216521	117	18	0	229853
Probe	2206	1900	43	17	0	4166
R2L	15889	201	216	18	0	16324
U2R	67	2	1	0	0	70
Overall	89204	220757	824	149	0	

References

[1] J. Blackard and D. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151, 2000.

[2] G. Cawley and N. Talbot. Manipulation of prior probabilities in support vector classification. In *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN '01.*

[3] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In D. Barbara and S. Jajodia, editors, *Applications of Data Mining in Computer Science*, chapter 4. Kluwer, 2002.

[4] C. Gathercole and P. Ross. Some training subset selection methods for supervised learning in genetic programming. 1994. http://citeseer.nj.nec.com/gathercole94some.html.

[5] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice-Hall, 2nd edition, 1999.

[6] H. G. Kayacik. Hierarchical self organizing map based ids on kdd benchmark. Master's thesis, Dalhousie University, 2003.

[7] R. King, C. Feng, and A. Shutherland. Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9(3):259–287, 1995.

[8] Ron Kohavi. Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207, 1996.

[9] T. Kohonen. *Self-Organizing Maps.* Springer, 1st edition, 1995.

[10] R. Kothari and V. Jain. Learning from labeled and unlabeled data using a minimal number of queries. In *IEEE Transactions on Neural Networks*, volume 14, pages 1496–1505. November 2003.

[11] Laboratory of Computer and Information Science, Neural Networks Research Centre, Helsinki University of Technology. http://www.cis.hut.fi/research/software.shtml.

[12] I. Levin. Kdd-99 classifier learning contest: Llsoft's results overview. *ACM SIGKDD Explorations*, 1(2):67–75, 2000.

[13] P. Lichodzijewski, A. N. Zincir-Heywood, and M. I. Heywood. Host-based intrusion detection using self-organizing maps. In *Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN '02.*

[14] N. Lightowler, C. T. Spracklen, and A. R. Allen. An introduction to modular map systems. *IEE Neural and Fuzzy Systems: Design, Hardware and Applications, Colloquium Digest*, 97/133:3/1–3/4, 1997.

[15] B. Pfahringer. Winning the kdd99 classification cup: Bagged boosting. *ACM SIGKDD Explorations*, 1(2):65–66, 2000.

[16] D. Song. A linear genetic programming approach to intrusion detection. Master's thesis, Dalhousie University, 2003.

[17] M.-C. Su and H.-T. Chang. Fast self-organizing feature map algorithm. *IEEE Transactions on Neural Networks*, 11(3):721–733, 2000.

[18] A. Ultsch and H. Siemon. Kohonen's self-organizing feature maps for exploratory data analysis. In *Proceedings of the International Neural Network Conference (INNC'90), Dordrecht, Netherlands*, pages 305–308. Kluwer, 1990.