SAAPS

Satellite Anomaly Analysis and Prediction System

Technical Note 3

Satellite Anomaly Prediction Module

Version 0.2

ESA/ESTEC Contract No. 11974/96/NL/JG(SC)

P. Wintoft

17 January 2000

Document status sheet

Technical Note 3, SAAPS-SAPM	
Version	Date
0.1	28 December 1999
0.2	17 January 2000

Contents

Chapter 1

Introduction

This document will describe models and prediction techniques that could be useful for the SAAPS.

Chapters 1.1 and 1.2 examines past work done in the predictions of satellite anomalies and energetic electron flux in the magnetosphere. Chapters 2 and 3 describes the neural networks and the fuzzy systems. Chapters 4 and 5 explores the prediction of electron flux and satellite anomalies.

1.1 Prediction of electron fluxes

There are a large number of papers on the subject of energetic electron fluxes in the magnetosphere. Surprisingly, there are only few papers that examine the prediction of the electron flux. [*Koons and Gorney,* 1991] developed a neural network to predict the daily average electron flux for energies > 3 MeV at geosynchronous orbit. The electron data was taken from the spectrometer for energetic electrons (SEE) on the (LANL?) 1982-019 satellite. The input to the network was a time delay line over the past 10 days of the daily sum Kp. The model was trained so that the day of the prediction was the same as the last day of the sum Kp, thus the model was trained to perform nowcasting. Then the model was tested to make one-day-ahead forecasts. The network model showed that the predictions were significantly more accurate than a linear filter. This model was extended to make predictions one day ahead and also included the electron flux itself at the input [*Stringer and McPherron,* 1993]. One-hour-ahead predictions of the GOES-7 electron fluxes has also been examined [*Stringer et al.,* 1996]. The input to the network was the Dst, Kp, the hourly average electron flux, and magnetic local time (MLT). [*Freeman et al.,* 1998] developed a neural network to predict the slope and intercept of the electron power law using Dst and local time as inputs. The intercept (B) and the slope (M) relates the electron energy (E, in keV) to the differential flux (F, electrons/cm^2 s sr keV)

according to

$$\log F = B + M \log E.$$

The power law is valid for electron energies 100 keV to 1.5 MeV. The electron data was taken from the (LANL?) 1984-129 satellite.

1.2 Prediction of satellite anomalies

To predict satellite anomalies introduces another level of difficulty as compared to predicting the space environment. The occurrence of an anomaly depends also on the design and age of the satellite, and different anomaly types have different origin.

[*Andersson et al.,* 1999] ...

Chapter 2

Artificial neural networks

2.1 Mathematical notation

The notation and mathematics follow as closely as possible as that used by [*Haykin,* 1994].

2.2 Multi-layer feed-forward NN

The multi-layer feed-forward neural network (MLFFNN) consists of several layers of neurons. The activation, i.e. the weighted sum of the inputs, for neuron j at layer l is

$$v_j^{(l)}(n) = \sum_{i=0}^{p} w_{ji}^{(l)} y_i^{(l-1)}(n). \tag{2.1}$$

The input at neuron i at layer $l-1$ for pattern n is $y_i^{(l-1)}(n)$. The activation is calculated by summing over all inputs (p neurons) multiplied with the weights $w_{ji}^{(l)}$. Then the activation is transformed to give the output at neuron j at layer l

$$y_j^{(l)}(n) = \varphi(v_j^{(l)}(n)), \tag{2.2}$$

where the activation function φ can be chosen in several ways. It depends on the layer and the problem type.

For hidden layers, we will always chose the hyperbolic tangent for the activation function

$$\varphi(v) = a \tanh bv = a\frac{1 - e^{-bv}}{1 + e^{-bv}}. \tag{2.3}$$

The constants a and b determine the amplitude and slope of the activation function. When $v \to \pm\infty$ then $\varphi \to \pm a$. The slope at $\varphi(0)$ is ab. In practice, a and b are often set to 1. However, it is interesting to examine when $b > 1$, especially for classification problems.

For the output layer we will either choose the tanh function or a linear function

$$\varphi(v) = bv. \tag{2.4}$$

The choice depends on whether we want to solve a curve fitting problem or a classification problem. For a curve fitting, or function approximation, problem it is generally better to chose a linear output function in order to avoid saturation at large values. In the case of a classification problem the tanh output function should be used as we wish to push to output to either false or true ($-a$ or $+a$).

In most cases it is sufficient to have one hidden layer. This type of network will solve any continuous valued function approximation problem [*Cybenko,* 1989] or any convex classification problem. A network with two hidden layers will solve any (continuous or discontinuous) function approximation problem or any (convex or non-convex) classification problem.

2.2.1 Error measures

To train a MLFFNN known input-output pairs must be available. If we call the desired output for output neuron j to be $d_j(n)$ then the error between the desired output and the network output becomes

$$e_j(n) = d_j(n) - o_j(n), \tag{2.5}$$

where $o_j(n) = y_j^{(L)}(n)$ and L is the output layer.

The standard error measure used in back-propagation learning is the summed squared error. The instantaneous sum of squared errors is

$$E(n) = \frac{1}{2}\sum_j e_j^2(n). \tag{2.6}$$

The sum over j includes all the neurons in the output layer. Also summing over all pattern n we get the sum of squared errors

$$E = \sum_{n=1}^{N} E(n). \tag{2.7}$$

Finally, we may also use the average of squared errors

$$E_{\text{av}} = \frac{1}{N}E. \tag{2.8}$$

Other error measures can also be used.

2.2.2 Back-propagation learning

The goal of the error-back-propagation algorithm is to minimize the error E by adjusting the weights w_{ji}. This is achieved by changing the weights so the that the error gradient $\partial E/\partial w_{ji}$ is negative. As the error is only known at the output layer, the errors have to be back-propagated through the preceding layers to be able to calculate an error gradient. The weights are updated according to

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \Delta w_{ji}^{(l)}(n), \tag{2.9}$$

where

$$\Delta w_{ji}^{(l)}(n) = \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n) + \alpha \Delta w_{ji}^{(l)}(n-1). \tag{2.10}$$

Here η is the learning coefficient and α is the momentum term. The local gradient $\delta_j^{(l)}(n)$ is

$$\delta_j^{(L)}(n) = e_j^{(L)}(n)\varphi'(v_j^{(L)}(n)) \tag{2.11}$$

for a neuron at the output layer L and

$$\delta_j^{(l)}(n) = \varphi'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) \tag{2.12}$$

for a neuron at hidden layer l. The prime function for a tanh activation function is

$$\varphi'(v) = ab(1 - \varphi^2(v)). \tag{2.13}$$

2.2.3 Back-propagation with adaptive learning coefficient

The standard back-propagation algorithm is slow. To speed things up the learning coefficient η can be varied during training. The idea is that each weight should have its own learning coefficient. The coefficient should increase when the error gradient $\partial E/\partial w_{ji}$ is negative, and decrease when the gradient is positive. This can be achieved with

$$\Delta \eta_{ji}(n) = \begin{cases} \kappa & \text{if} S_{ji}(n-1) D_{ji}(n) > 0 \\ -\beta \eta_{ji}(n) & \text{if} S_{ji}(n-1) D_{ji}(n) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

where $S_{ji}(n-1)$ and $D_{ji}(n)$ are defined as, respectively

$$D_{ji}(n) = \frac{\partial E(n)}{\partial w_{ji}(n)} \tag{2.15}$$

and

$$S_{ji}(n-1) = (1-\xi) D_{ji}(n-1) + \xi S_{ji}(n-1) \tag{2.16}$$

where ξ is a positive constant. The procedure is called the delta-bar-delta learning rule [*Jacobs,* 1988].

2.2.4 Marquardt-Levenberg algorithm

Both the standard backpropagation (BP) and the adaptive learning rate BP algorithms may demand long training times. A more efficient training procedure is the Marquardt-Levenberg backpropagation (MBP) algorithm [*Hagan and Menhaj,* 1994]. MBP is an approximation to the Newton method

$$\Delta \mathbf{w} = -[\nabla^2 E(\mathbf{w})]^{-1} \nabla E(\mathbf{w}), \tag{2.17}$$

where $\mathbf{w}$ are the weights and the error function E is the sum of squared errors (Eq. 2.7). With this error function the terms in Eq. 2.17 can be rewritten using the Jacobian matrix. Neglecting higher order terms the Newton method becomes the Gauss-Newton method. MBP is a modification of the Gauss-Newton method and can be expressed as

$$\Delta \mathbf{w} = [J^T(\mathbf{w}) J(\mathbf{w}) + \mu I]^{-1} J^T(\mathbf{w}) \mathbf{e}(\mathbf{w}). \tag{2.18}$$

The parameter μ is multiplied by a factor β whenever a step would result in an increase of the error function E, and divided by β whenever E decreases. When μ is large the algorithm becomes the steepest descent with learning rate $\eta = 1/\mu$, and when μ is small the algorithm becomes the Gauss-Newton.

The Jacobian matrix is

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_1} & \frac{\partial e_2(\mathbf{w})}{\partial w_1} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_1} \\ \frac{\partial e_1(\mathbf{w})}{\partial w_2} & \frac{\partial e_2(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_1(\mathbf{w})}{\partial w_n} & \frac{\partial e_2(\mathbf{w})}{\partial w_n} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_n} \end{bmatrix}. \tag{2.19}$$

The number of columns N is equal to the product of the number of training examples Q and the number of layers L, thus $N = Q \times L$. The number of rows n is equal to the total number of weights in the network.

The advantage of the MBP is the much reduced training time. However, it also demands more memory to store the elements of the Jacobian matrix. We see that the size of the matrix depends both on the number of training examples and the total number of weights.

2.3 Radial basis function neural networks

The radial basis function neural network (RBFNN) is also a feed forward network, however, the hidden layer neurons use local transformation functions instead of the global functions used in the MLFFNN. The network has the following expression

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi(\|\mathbf{x} - \mathbf{x}_i\|), \tag{2.20}$$

where $\mathbf{x}_i$, $i = 1, 2, \ldots, N$, are the centres of the radial basis functions φ. The norm $\|\cdot\|$ is taken as the Euclidean length. The activation function is the Gaussian function

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right), \tag{2.21}$$

where σ is the width of the function.

2.4 Normalization

2.5 Weight initialization

[*Nguyen and Widrow,* 1990]

2.6 Training set selection

2.6.1 Classification problem

To train the network we need to carefully select the examples that should go into the training set. Ideally we would like to find the decision boundary in the input space that separates the two classes from each other (e.g. anomaly and non-anomalies). According to [*Swingler,* 1996], it is not desirable to have the distribution of training examples equal to the the distribution of real world examples, as the class with more training examples will dominate the training. As an example we know that the Meteosat-3 satellite experienced an anomaly day in about 20% of the time. Thus by always saying that we will have no anomalies we would be correct in 80% of the time. A network trained on such a set will thus bias its performance to predict the no-anomaly events. This will be further examined in Chapter 5.

The goal of the training algorithm is to minimize the error between the desired output and the network output. The measure used is most often the summed squared error (Eq. 2.7). Assume we have two classes, class A and class B. The total number of class A examples is n_A and the total number of B examples is n_B. Then the total number of examples is

$$n = n_A + n_B \tag{2.22}$$

and the fraction of examples of each class is

$$\begin{array}{ll} f_A = & \frac{n_A}{n} \\ f_B = & \frac{n_B}{n} \end{array}, \tag{2.23}$$

where $f_A + f_B = 1$. Assume also that a fraction c_A of the examples in class A are correctly classified, and c_B of the examples in class B. It should be

noted that $c_A \leq 1$ and $c_B \leq 1$, and that $c_A + c_B \leq 2$. The fraction of correct classification then becomes

$$\begin{aligned} c &= f_A c_A + f_B c_B & (2.24)\\ &= f_A c_A + (1 - f_A) c_B. & (2.25) \end{aligned}$$

This should now be compared with the summed squared error defined in Eq. 2.7. Rewriting Eq. 2.7 in terms of the individual contributions from the two classes we get

$$E = E_A + E_B, \tag{2.26}$$

where

$$\begin{aligned} E_A &= \tfrac{1}{2} \textstyle\sum_{n \in n_A} (d(n) - y(n))^2 \\ E_B &= \tfrac{1}{2} \textstyle\sum_{n \in n_B} (d(n) - y(n))^2 \end{aligned} \quad . \tag{2.27}$$

If we assume that we have a tanh transfer function at the output layer then the squared error $(d(n) - y(n))^2$ is either 0 for a correct classification or 4 for an incorrect classification. Thus if all A classification are wrong then $E_A = 1/2 n_A 4 = 2n_A$. Thus Eq. 2.26 can be written as

$$\begin{aligned} E &= E_A + E_B & (2.28)\\ &= 2n_A(1 - c_A) + 2n_B(1 - c_B) & (2.29)\\ &= 2n[f_A(1 - c_A) + f_B(1 - c_B)] & (2.30)\\ &= 2n[f_A(1 - c_A) + (1 - f_A)(1 - c_B)] & (2.31)\\ &= 2n[f_A + (1 - f_A) - f_A c_A - (1 - f_A) c_B] & (2.32)\\ &= 2n[1 - \{f_A c_A + (1 - f_A) c_B\}]. & (2.33) \end{aligned}$$

The goal of the training algorithm is to minimize E which, from Eq. 2.33, is identical to maximizing the total number of correct classifications c. Thus, depending on how we choose f_A it will effect the maximum value of c, and also the values of c_A and c_B.

2.6.2 Continuous valued problem

2.7 Training strategies

2.7.1 Training, validation and test

The data set that has been created fro the development of the neural network should be divided into three subsets: a training set, a validation set, and a test set. The training set is used to train the network, i.e. the process when the weights are changed so that the error is minimized. The validation set is used to determine the optimal network with respect to the number of hidden neurons and the selection of input parameters. It can also be used to decide when to stop training. Finally, when the optimal network has been selected

it can then be tested on the test set to find the performance that can be expected in a real situation. The three sets should not have any overlapping examples. The sets should also have similar statistics.

The three sets should contain a sufficient number of examples so that the network can be properly trained, validated and tested. The number of examples needed is problem related; a small simple network needs fewer examples than a large complex network.

2.7.2 Training with minimum set size

Sometimes it is not possible to divide the data set into the three sets because the set is to small. Instead one can use only a training set and then estimate the size of the network that will generalize well given the size of the data set and the performance of the network. According to [*Baum and Haussler,* 1989] a binary classifying network (output 0 or 1) with one hidden layer will almost certainly provide good generalization if the number of examples in the training set N satisfies

$$N \geq \frac{32W}{\varepsilon} \ln \frac{32M}{\varepsilon} \tag{2.34}$$

where W is the number of weights in the network and M is the number of hidden neurons. If ε is small then the network performs very well on the training set. If the training set is small (N small) it is probable that the network simply has memorized the data and will thus generalize poorly. On the other hand if N is large it is likely that the network has found the underlying general relationship. Equation 2.34 represents the worst-case formula for estimating the training set size. In practice it seems that it is sufficient to keep the first term and dropping the constants [*Haykin,* 1994] so that we get

$$N > \frac{W}{\varepsilon}. \tag{2.35}$$

We can now illustrate equations 2.34 and 2.35 with the anomalies on Meteosat-3. During its operation the satellite experienced about 700 anomalies. If we assume a network with 10 input units (e.g. $\sum K_p$ over 10 days) and 10 hidden units then $M = 10$ and $W = (10 + 1) \times 10 + 10 + 1$. The minimum number of examples needed as a function of $\varepsilon/2$ would vary according to Figure 2.1. Using the criterion from Equation 2.34 we see that N should be much larger than the 700 available examples, thus it can not be guaranteed that the network will generalize. Using the relaxed criterion (Eq. 2.35) we now see that the Meteosat-3 anomaly set is sufficiently large to be used for training when the fraction of errors ($\varepsilon/2$) are larger than 0.1.

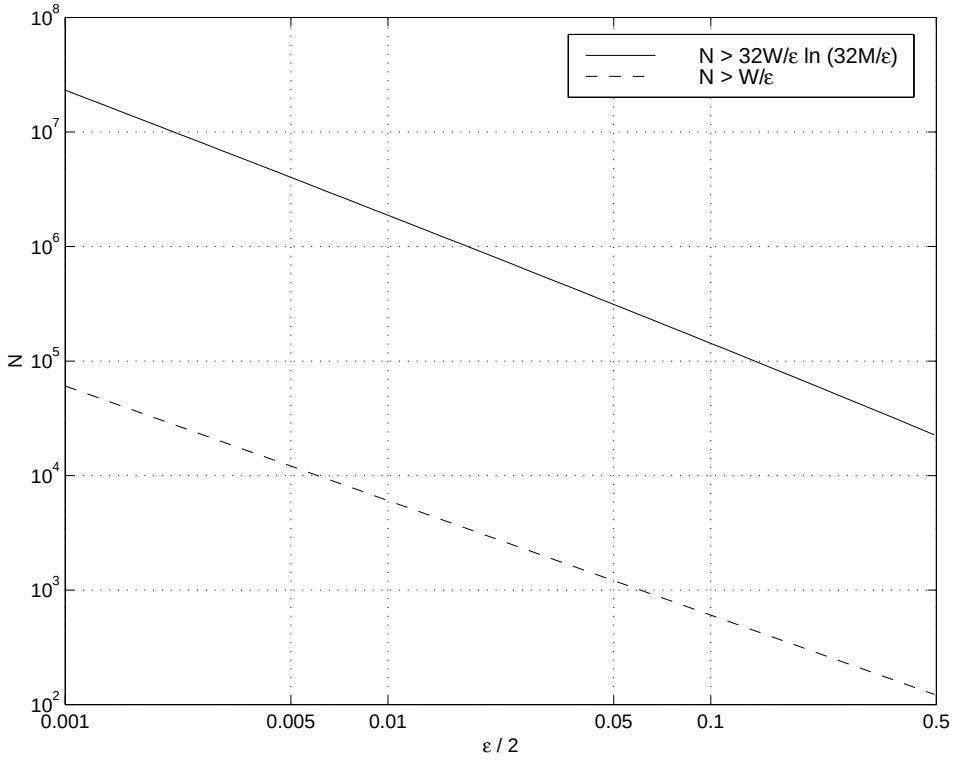

Figure 2.1: The minimum number of examples (N) as a function of the fraction of errors ($\varepsilon/2$) for a network with 10 input units and 10 hidden units.

Chapter 3

Fuzzy systems

Chapter 4

Prediction of the electron flux from solar wind data

As described in section 1.1, past attempts to predict the electron flux at geosynchronuous orbit has always used the local electron flux and/or geomagnetic indices (Kp, Dst). The emphasis has been on high energies (MeV) which are believed to give internal charging.

As it is the solar wind that drives magnetospheric activity it is natural to develop models to predict the electron flux using solar wind data. As the electron flux can vary by several orders of magnitude within 24 hours the time resolution of the predictions should be better than one day. If a time resolution of one hour is used the diurnal variation is captured. At the same time we avoid difficulties associated with substorm dynamics and the evolution of the solar wind from L1 to the Earth.

As the satellite measures the electron flux at a single point and at the same time moves in the 24-hour orbit around the Earth, it is not possible to distinguish between spatial and temporal variations. This problem can be solved in two ways: (1) The input is the solar wind data with 24 different networks predicting the flux in each local time sector; (2) The input is the solar wind data and the local time, and only one network predicting the flux in all local time sectors. It is difficult to say which method will work best. The first method produces simpler networks as each network only has to model one time sector. However, the training procedure is more complicated as 24 networks need to be trained. The number of available training examples will also be reduced by a factor of 24. The second method will produce a more complex network as it also has to model the local time variation. The training procedure is simpler, only one network, but the final prediction accuracy might be poorer as a collection of simpler specialized networks usually perform better than one general network. Both approaches shall be examined. The first approach is also similar to the hybrid neural network [*Haykin,* 1994].

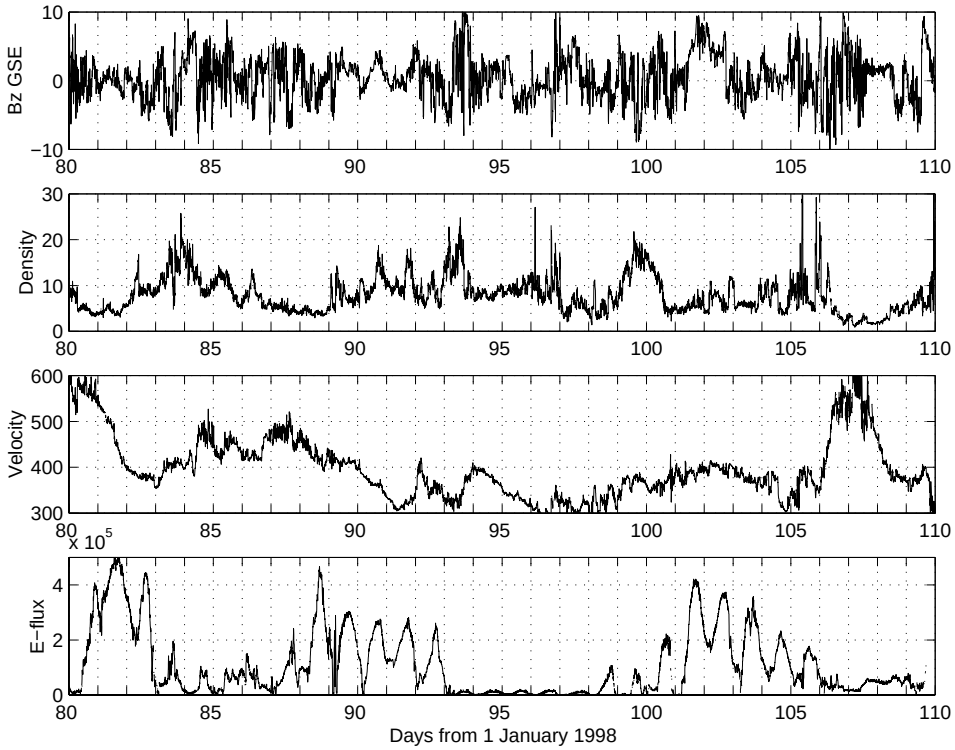

Figure 4.1: The solar wind magnetic field (B_z), density, velocity, and the > 0.6 MeV electron flux over 30 days in 1998.

4.1 GOES > 0.6 MeV

4.2 GOES > 2 MeV

4.3 LANL SOPA 50-75 keV

4.4 LANL SOPA 75-105 keV

4.5 LANL SOPA 105-150 keV

4.6 LANL SOPA 150-225 keV

4.7 LANL SOPA 225-315 keV

4.8 LANL SOPA 315-500 keV

4.9 LANL SOPA 500-750 keV

Chapter 5

Prediction of satellite anomalies

The prediction of satellite anomalies is a binary yes/no-problem, or a classification problem. This can be compared to the prediction of the electron flux which is a continuous valued problem. The prediction of an anomaly is either completely correct or completely wrong.

Essentially one would like to find a parameter space that can be separated into two classes that corresponds to anomaly and no-anomaly, respectively.

5.1 Daily predictions

5.1.1 Meteosat-3 anomalies predicted using $\sum K_p$

As the daily average electron flux has been predicted using $\sum K_p$ it is natural to develop an anomaly prediction model also using $\sum K_p$. This was examined by [*Wu et al.,* 1998]. The analysis will be repeated here, however, the definition of anomaly and non-anomaly will be slightly different. If one or several anomalies occur during a day, then that day is an anomaly day. If no anomalies occur during a day, that day is considered a non-anomaly day.

During the satellites 7 year lifetime (1988-1995) it experienced 724 anomalies. The anomalies were spread over 497 days, which should be compared to the total number of 2678 days for this period. The fraction of days with anomalies were thus 0.19. On average there were a day with anomalies every 5.4 days.

Using superposed epoch analysis one can discover trends in K_p that precede the anomalies. Figure 5.1 shows how the 3-hour K_p starts increase 6 days before the anomaly and reaches the maximum value one day ahead. However, one should also note the large variation of K_p for individual events as indicated by the standard deviation (dashed lines).

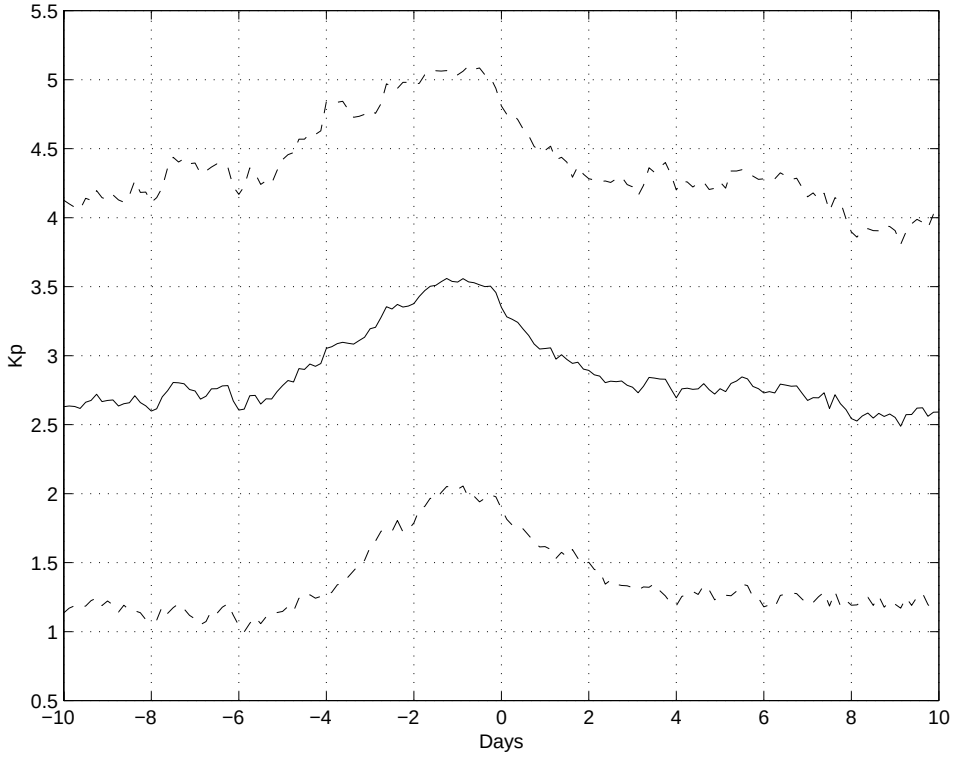

Figure 5.1: The 3-hour Kp superposed on anomaly events from Meteosat-3. The solid line shows the superposed values and the dashed lines are the standard deviation.

As was shown in Section 2.6 the minimization of the summed squared error is identical to maximizing the fraction of correct predictions c. The problem is to select a suitable training set as the number of days with anomalies is smaller than the number of days without anomalies.

To examine the effect of choosing the training and test sets with different distributions we construct a network with 8 input units, 3 hidden units and 1 output unit. The inputs are 8 days of $\sum K_p$ and the output is whether there is an anomaly or not the next day. This network size produced similar errors on both the training set and the test set. A larger network always led to very small errors on the training set and larger errors on the test set. The total number of training examples has been fixed to $n^{tr} = 497$ and then the fraction of anomalies in the training set f_A^{tr} has been varied over $0.1, 0.2, 0.3, 0.4, 0.5$. This means that the number of anomalies in the training set n_A^{tr} has been $49, 99, 149, 198, 248$. Then the networks have been tested on the test set where also the fraction of anomalies f_A^{ts} have been varied over $0.1, 0.2, 0.3, 0.4, 0.5$. The result is shown in Figure 5.2. When we have a balanced training set ($f_A^{tr} = 0.5$) then the fraction of correct classifications is close to constant ($c^{ts} \approx 0.65$) independent of the test set distribution (f_A^{ts}). When the training set is unbalanced ($f_A^{tr} < 0.5$) makes c^{ts} dependent on the test set distribution, where c^{ts} increases for decreasing f_A^{ts}. The result is repeated in Figure 5.3 where each panel shows the variation of c^{ts} as a function of f_A^{ts} for constant f_A^{tr}. The solid-circle curves are the slices for constant f_A^{tr} from Figure 5.2. Also shown in the figure are the fractions of correct classifications for anomalies (c_A^{ts}, dash-square curve) and no-anomalies (c_B^{ts} dot-diamond curve), respectively. For the balanced training set $c^{ts} \approx c_A^{ts} \approx c_B^{ts}$. This we can understand from the training procedure which minimizes E (Eq. 2.7) or equivalently maximizes

$$c^{tr} = f_A^{tr} c_A^{tr} + f_B^{tr} c_B^{tr}. \tag{5.1}$$

As $f_A^{tr} = f_B^{tr}$ both c_A and c_B will be maximized with equal weights. With an unbalanced training set c_A and c_B will be maximized unevenly, and if $f_A^{tr} < f_B^{tr}$ then the training algorithm will bias toward maximizing c_B. This is seen in Figure 5.3 and is mostly pronounced for the case when $f_A^{tr} = 0.1$.

5.2 Hourly predictions

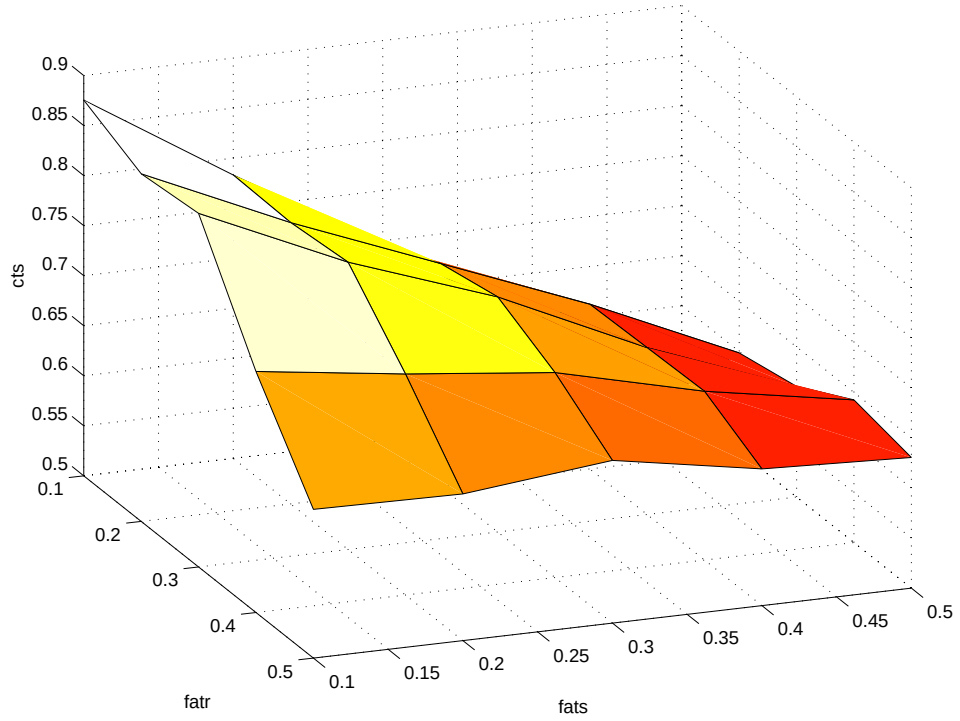

Figure 5.2: The fraction of correct predictions on the test set (cts) as a function of the fraction of anomalies in the training set (fatr) and the fraction of anomalies in the test set (fats).

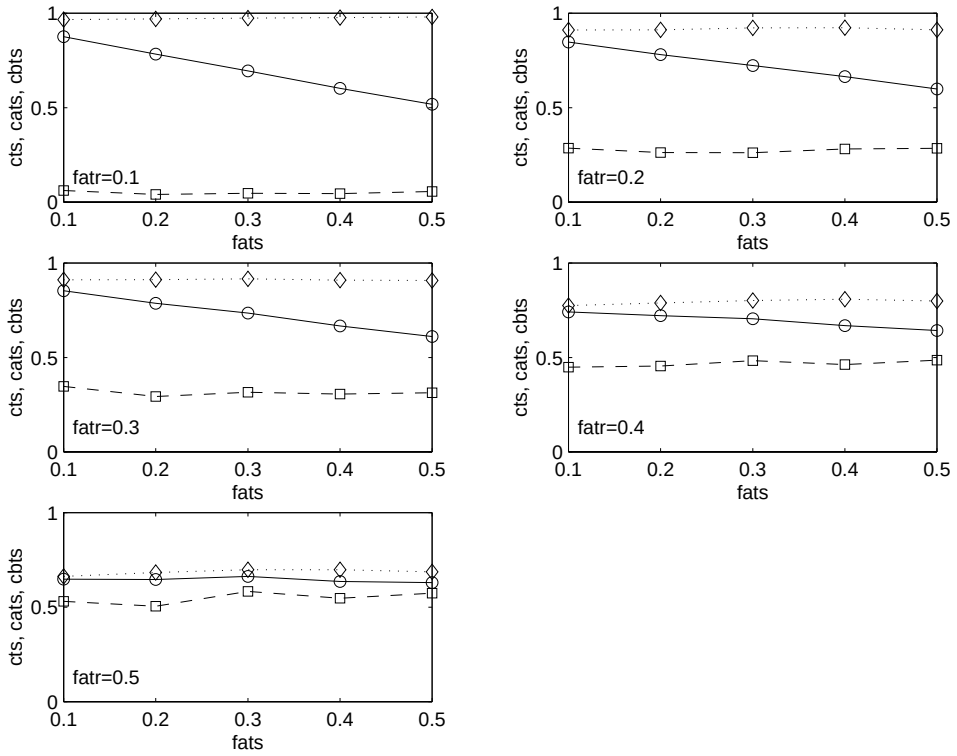

Figure 5.3: The fraction of correct predictions on the test set (cts) as a function of the fraction of anomalies in the test set (fats) for different values of the fraction of anomalies in the training set (fatr) (solid curve). The fraction of correct predictions for the anomalies (cats, dashed curve) and the no-anomalies (cbts, dotted curve), respectively, are also shown.

References

Andersson, L., L. Eliasson, and P. Wintoft, Prediction of times with increased risk of internal charging on spacecraft, *Workshop on space weather,* ESTEC, Noordwijk, WPP-155, 427–430, 1999.

Baum, E.B., and D. Haussler, What size net gives valid generalization?, *Neural Computation 1,* 151–160, 1989.

Cybenko, G., Approximation by superpositions of a sigmoidal function, *Math. Control Signals Systems* 2, 303–314, 1989.

Hagan, M.T., and M. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Trans. Neural Netwworks, 5*, 989–993, 1994.

Haykin, S., *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, Inc., New York, 1994.

Jacobs, R.A., Increased rates of convergence through learning rate adaption, *Neural Networks* 1, 295–307, 1988.

Freeman, J.W., T.P. O'Brien, A.A. Chan, and R.A. Wolf, Energetic electrons at geostationary orbit during the November 3-4, 1993 storm: Spatial/temporal morphology, characterization by a power law spectrum and, representation by an artificial neural network, *J. Geophys. Res., 103,* 26,251–26,260, 1998.

Koons, H.C., and D.J. Gorney, A neural network model of the relativistic electron flux at geosynchronous orbit, *J. Geophys. Res., 96,* 5,549–5,556, 1991.

Nguyen, D., and B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, in *Proc. IJCNN,* 3, 21–26, 1990.

Stringer, G.A., and R.L. McPherron, Neural networks and predictions of day-ahead relativistic electrons at geosynchronous orbit, *Proceedings of the International Workshop on Artificial Intelligence Applications in Solar-Terrestrial Physics, Lund, Sweden, 22-24 September 1993,* Edited by J.A. Joselyn, H. Lundstedt, and J. Trolinger, Boulder, Colorado, 139–143, 1993.

Stringer, G.A., I. Heuten, C. Salazar, and B. Stokes, Artificial neural network (ANN) forecasting of energetic electrons at geosynchronous orbit, in *Radiation Belts: Models and Standards, Geophys. Monogr. Ser.,* vol 97, edited by J.F. Lemaire, D. Heynderickx, and D.N. Baker, AGU, Washington, D.C., 291–295, 1996.

Swingler, K., *Applying neural networks: a practical guide*, Academic Press Ltd, London, 1996.

Wu, J.-G., H. Lundstedt, L. Andersson, L. Eliasson, and O. Norberg, Spacecraft anomaly forecasting using non-local environment data, *Study of plasma and energetic electron environment and effects (SPEE),* ESTEC Technical Note, WP 220, 1998.