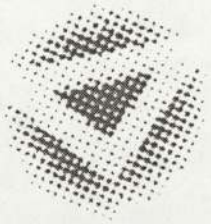
Neural Networks for Modelling Wind Vectors

GUILLAUME RAMAGE

Master of Science (by Research)
in
Pattern Analysis and Neural Networks

Supervisor: Dr Dan Cornford



ASTON UNIVERSITY

Submission date: September 1998

Aston University

Neural Networks for Modelling Wind Vectors

Guillaume Ramage

Master of Science (by Research)
in
Pattern Analysis and Neural Networks, 1998

Thesis Summary

The ERS-1 satellite was launched in 1991. It carries a scatterometer with three antennae which measure the reflected radar power from the surface of the Earth. This backscatter is due to the reflection of the micro-wave radar beam from small ripples on the surface of the ocean which are generated by instantaneous winds. The resulting measurement triplet can be used to infer wind vectors.

The implementation of a forward model which maps wind vectors to radar backscatter is addressed here, applying techniques from the field of neural networks. An empirical approach is adopted here. The neural networks are trained with wind data from the European Centre for Medium-Range Weather Forecasting in which high wind speeds occur. The poor quality of the models obtained demonstrates that the noise in this input data cannot be neglected. A Bayesian framework is then adopted to account for this noise. Compared to existing reference models, the fit of the model in target space is improved, especially at high wind speeds which are of greatest interest for meteorological studies. Although the inversion of the model is not implemented, its potential accuracy is higher than existing models.

Keywords: Wind vectors retrieval, ERS-1 satellite, probabilistic models, non-linear regression, neural networks, input uncertainty

Contents

List of Figures

Acknowledgements

I would like to thank Dr Ian Nabney for his useful comments during this project.

I would also like to thank Dr Dan Cornford for his invaluable help and for the stimulating environment he provided.

I am also grateful to all students at the NCRG for their help and more specifically to David Evans for the useful exchanges we had.

Chapter 1

Introduction

1.1 Technical background

The European Space Agency (ESA) launched its first Earth Remote-Sensing satellite ERS-1 on 17 July 1991. It carries many instruments, including the Advanced Microwave Instrument (AMI) which is a combined wind and wave scatterometer. This scatterometer measures the return radar power from three antennae that form a swathe to the right side of the satellite ground track (Figure 1.1). This backscatter is generated by the surface of the ocean. As the on-board microwave radar operates at 5.3 GHz (C-band VV polarisation), the backscatter is mostly due to the in-phase reflection on small ripples of around 5 cm wavelength which are generated in turn by instantaneous surface winds.

The three antennae point at an azimuth of 45, 90 and 135° to the satellite heading for the fore, mid and aft beam respectively. They sweep a 500 km wide swathe along the track of the satellite. 19 cells are sampled every 25 km across this swathe. As the dimension of these cells is 50 by 50 km, they overlap to some extent. The satellite executes its polar orbit in 100 minutes and travels forward at 6.7 km/s. So each cell is swept in turn by the three beams in a few minutes, depending on the position across the swath. This results in a triplet of measured backscatter $\boldsymbol{\sigma}^o = (\sigma_f^o, \sigma_m^o, \sigma_a^o)$. This information, together with the incidence angles of the beams, can then be used to determine the wind vectors over the ocean.

Obtaining wind vectors over the ocean is important to Numerical Weather Prediction

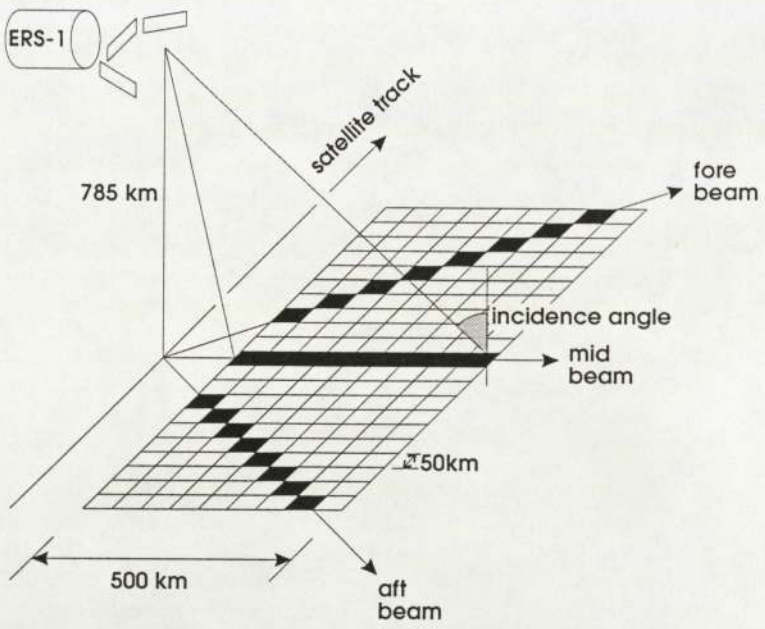


Figure 1.1: The ERS-1 scatterometer geometry. Although this is not represented here, the assumption of a flat sea surface is wrong.

(NWP) since the ability to produce a forecast of the future state of the atmosphere depends critically on knowing the current state accurately. It is also important for studying meteorological phenomena such as fronts or cyclones at a high resolution, and for studying basin-wide ocean circulation models.

1.2 Different techniques of wind retrieval

1.2.1 Existing approaches

Many algorithms have already been developed for the retrieval of wind vectors from scatterometer data. The function we are really interested in is $(\boldsymbol{\sigma}^o \mapsto (u, v))$. But this is a multi-valued function, as the same $\boldsymbol{\sigma}^o$ vector can be measured for opposite wind directions. This is due to the similarity of the shape of the ripples moving in opposite directions. Hence this function is called an *inverse* model. By contrast, the forward model $((u, v) \mapsto \boldsymbol{\sigma}^o)$ is unimodal. Almost all existing algorithms are built on the inversion of a forward model. It is often assumed that the three antennae of the satellite are equivalent. A model with

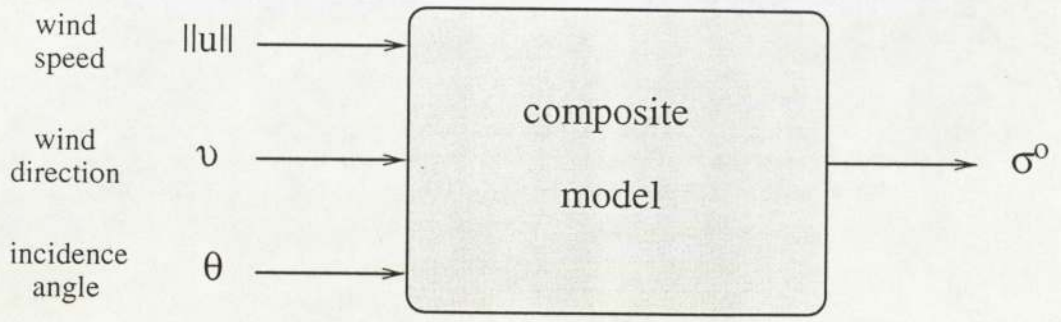


Figure 1.2: Sketch of a composite model. It has one output only. Whether the inputs are (u, v) or $(\|\boldsymbol{u}\|, \vartheta)$ only plays a role when the model is trained.

one output only is used for all antennae, it is then called a composite model (see sketch in Figure 1.2). It is run three times to get a $\boldsymbol{\sigma^o}$ triplet. The most natural representation of the composite model is shown in Figure 1.3: σ^o is a function of (u, v) with parameter θ.

Figure 1.3 shows CMOD4 and nn2cmod (one of the models developed during this thesis) for a speed range of 3-26 m/s (CMOD4 is only valid up to 15 m/s) and for incidence angles $\theta = 18°$ (top funnel), 28.6°, 37.7° and 45.4° (lowest funnel) which correspond to mid beam incidence angles for tracks 1, 7, 13 and 19 respectively. The colours represent constant speeds and the straight line indicates up-wind direction. This is a good illustration of what can be read in (Rufenach, 1995): *The cross-sectional slope varies from about zero dB / m/s at high wind speeds* $\|\boldsymbol{u}\| = 18$ *m/s and small incidence angles* $\theta = 20°$ *to about 1.3 dB / m/s at low wind speeds* $\|\boldsymbol{u}\| = 3$ *m/s and large incidence angles,* $\theta = 55°$.

Very recently (Janssen *et al.*, 1998) proposed a physically based theoretical ocean backscatter model (called VIERS-1). Unlike other models presented below, the expression of this model is very complicated and it also makes use of a wave prediction model. Therefore a direct comparison of its performance with the models which will be presented here is not possible as it cannot be easily implemented.

However most of the existing models are obtained empirically. The models presented in this document are also built from an empirical approach.

Forward models can be classified according to two criteria: the source of data used for tuning and whether a prior hypothesis of a functional form is made.

The following models appear in (Rufenach, 1997), where a comparison of their perfor-

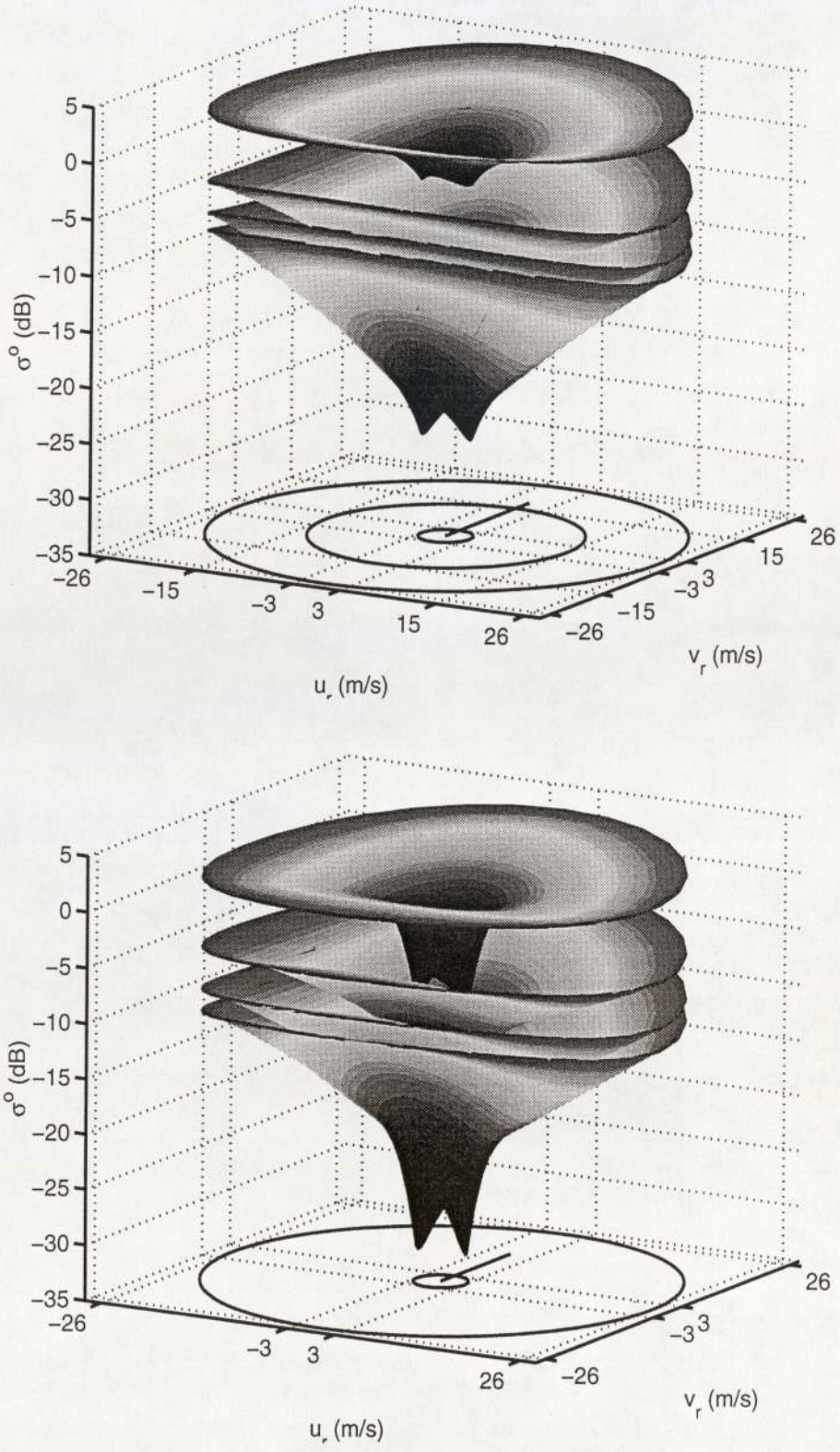


Figure 1.3: Most natural representation of the composite forward model: CMOD4 (top) and nn2cmod (bottom).

mances is proposed on the basis of a comparison between predicted wind vectors and collocated winds from buoy measurements:

- The CMOD4 function was trained using analysis winds from the European Centre for Medium-Range Weather Forecasts (ECMWF). It uses a functional form (see next section).
- The CMOD-IFR2 function was trained using a mixture of ECMWF winds and winds from buoy measurements. It also has a functional form.
- The model from Oregon State University doesn't use any functional form, it is tuned with NWP winds.

CMOD4 and CMOD-IFR2 are presented in Section 1.3.

1.2.2 NEUROSAT overview

In (Thiria *et al.*, 1993), artificial data was used to train several neural networks for the inverse model. In this work, σ^o values from several contiguous cells were gathered to facilitate the wind vector retrieval. More recently, a forward model was obtained using a Multi Layer Perceptron neural network (Mejia *et al.*, 1998). The data used in this study was filtered with the help of this group at the Laboratoire d'Océanographie Dynamique et de Climatologie (LODYC, Université de Paris 6, France). The wind retrieval model for the NSCAT scatterometer (Mejia *et al.*, 1998) has also been studied by this group.

Other studies carried out at the Neural Computing Research Group at Aston University include the design of an inverse model where the tasks of wind direction and speed retrievals were split (Cornford *et al.*, 1997). Also in (Nabney and Bishop, 1995b) and (Nabney and Bishop, 1995a), some techniques are proposed for the retrieval of wind direction from σ^o measurements using mixtures of wrapped Normal densities.

Alongside this thesis, MSc student David Evans is also developing an inverse model using mixture density networks. His inverse model is used during this study.

A general overview of the NEUROSAT project is given in (Cornford and Nabney, 1998b).

1.3 Reference forward models

Three existing models are presented in this section: CMOD4, CMOD-IFR2 and NN-GMF.

1.3.1 CMOD4 and CMOD-IFR2

CMOD4 is a geophysical forward model. Its functional form is based on the use of a truncated Fourier series. It was developed at the European Space Agency (ESA). Its functional form is defined in (Offiler, 1994) as:

$$\sigma^o_{\text{lin}} = B_0(1 + B_1 \cos\vartheta + B_3 \tanh(B_2) \cos 2\vartheta)^{1.6} \tag{1.1}$$

where B_0, B_1, B_2, B_3 are complicated functions of the wind speed $\|\boldsymbol{u}\|$, the relative wind direction ϑ, and the beam incidence angle θ (symbols conventions are listed in Appendix A) with 18 tunable coefficients overall. In this form, σ^o is in linear measure. The conversion to decibels is given by:

$$\sigma^o_{dB} = 10 \log_{10} \sigma^o_{lin} \tag{1.2}$$

The CMOD4 function was tuned using a sample of 20,000 wind vectors derived from ECMWF analysis, with a maximum likelihood estimation (MLE) method. Some of the parameters and thresholds were determined by empirical methods. Although CMOD4 was developed four years ago, it is still used operationally in several meteorological offices across Europe. Articles written by its authors (Stoffelen and Anderson, 1997b; Stoffelen and Anderson, 1997a; Stoffelen, 1998) are the most relevant in this work.

Another geophysical model called CMOD-IFR2 was developed at the Institut Français de la Recherche pour l'Exploitation de la Mer (IFREMER). Its functional form is close to CMOD4's. It is defined in (Maroni *et al.*, 1995) as:

$$\sigma^o_{\text{lin}} = B_0(1 + B_1 \cos\vartheta + B_3 \tanh(B_2) \cos 2\vartheta) \tag{1.3}$$

This model is used at the CERSAT (Centre ERS d'Archivage et de Traitement) to find outliers in the measured $\boldsymbol{\sigma^o}$ triplets. CMOD4 and CMOD-IFR2 are both composite mo-

dels: they are single-output functions so they implicitly assume that the three antennae of the satellite are equivalent.

1.3.2 NN-GMF: a neural network

The NN-GMF forward model (Mejia *et al.*, 1998) was developed at the Laboratoire d'Océanographie Dynamique et de Climatologie very recently. It is a Multi Layer Perceptron neural network with five hidden units. It has 5 inputs: the wind speed $\|\boldsymbol{u}\|$, $\sin(\vartheta)$ and $\cos(\vartheta)$ for the relative wind direction and $\cos(\theta)$ and $\sin(\theta)$ for the incidence angle. Unlike the two models above, NN-GMF contains no a priori hypothesis[1] on the relationship between $\boldsymbol{\sigma^o}$ and (u, v, ϑ). This model was trained on ECMWF samples where priority was given to speeds in the range 3 to 15 m/s. Wind speeds higher than 18 m/s were not present.

1.3.3 Wind retrieval using CMOD4

The CMOD4 function gives a mapping $((\|\boldsymbol{u}\|, \vartheta, \theta) \rightarrow \sigma^o)$, that is, it is a forward model. However, the mapping we need in order to determine the wind vector from scatterometer data is $(\boldsymbol{\sigma^o}, \theta) \rightarrow (u, v)$, so CMOD4 is inverted with the help of lookup tables. The wind is obtained from the following algorithm (personal communication from Dave Offiler):

1. A first LUT gives sets of coefficients vs direction and incidence angle, so that given a σ^o value, we get $\|\boldsymbol{u}\| = LUT(\sigma^o, \vartheta, \theta)$. We apply this LUT at regular samples of direction to form a set of candidate ($\|\boldsymbol{u}\|$,ϑ) for $\vartheta \in [0, 360]$ at intervals of 15°.

2. Then for each direction, the root mean square error (rms) of measured σ^o against σ^o value computed by CMOD4. Local minima are found in sampled rms space, and some interpolation is employed to find the actual minima, say ϑ_i (i can go up to 3 or 4).

3. The LUT is used again to estimate each corresponding $\|\boldsymbol{u}\|_i$. We now have to remove the ambiguity of these sets of ($\|\boldsymbol{u}\|_i$,ϑ_i).

[1]The only (weak) assumption is that the relationship is infinitely continuously differentiable.

4. Select the set of $(\|\boldsymbol{u}\|_i,\ \vartheta_i)$ which is the closest to Numerical Weather Prediction (NWP) background winds (3-8 hrs forecast).

5. A median filter over contiguous tracks is applied to confirm or else update the result of the previous step.

A very different method is employed for CMOD-IFR2, it is fully explained in (Maroni *et al.*, 1995).

1.4 Aims of this project

One of the aspects this report investigates is the design and implementation of a probabilistic forward model, $P(\boldsymbol{\sigma}^o \mid u, v, \theta)$, relating backscattered radiation to wind vectors. Existing models such as CMOD4 only give the expectation $E[\boldsymbol{\sigma}^o \mid u, v, \theta]$. So currently, when a conditional or unconditional probability distribution of $\boldsymbol{\sigma}^o$ is required, it is implicitly assumed that the error distribution is spherical. Such a probabilistic model can be used in a Bayesian framework together with a wind field prior $P(U, V)$ to obtain wind fields, (U, V), from $\boldsymbol{\Sigma}^o$ values in a scene:

$$\begin{aligned} P(U, V \mid \boldsymbol{\Sigma}^o) &= \frac{\prod_i P(\boldsymbol{\sigma}_i^o \mid u_i, v_i) P(U, V)}{\prod_i P(\boldsymbol{\sigma}_i^o)} \\ &\propto \left[\prod_i P(\boldsymbol{\sigma}_i^o \mid u_i, v_i) \right] P(U, V) \end{aligned}$$

where i designates the ith cell in a scene.

1.5 About the layout of this document

As the progression in this work was mainly linear in time (as opposed to divided in several branches), it is presented in a rather chronological order, so the progression between chapters is preserved. It is important to note that some of the conclusions which are drawn in Chapters 2 and 3 are not correct. This is due to the presence of input noise in the data. The reader may prefer to jump straight to Section 5.2 to have an overview of the effects of input noise in this study.

A general presentation of the data used is proposed in Chapter 2. Some neural networks are trained in the usual framework of non-linear regression. Some more complicated neural networks are gathered in Section 3, including a "hybrid" model whose properties are chosen to compensate for some faults of the other networks, according to the graphical representations which are actually introduced in Chapter 4. It is shown in Chapter 5 that input noise cannot be neglected. The training method is modified in consequence. Further validation can then be undertaken (Chapter 6).

Chapter 2

Early work: modelling the expectation

2.1 Data

In order to build an empirical mathematical model relating wind vectors to satellite measurements, one first needs to collect information about the instantaneous wind vectors for areas of the sea which match the cells on the satellite track (50 by 50 km). There are two distinct sources for this information: ground measurements and data from Numerical Weather Prediction (NWP) models.

2.1.1 Ground measurements

There exist different ways of measuring the wind on the sea: buoys, ships, and aircraft are the most widely used. They give sparse (local) information about the wind vectors, at a lower resolution than the cells on the satellite track. In 1991 the RENE campaign (Offiler, 1994) was carried out so as to remedy to this problem. This campaign was intended to gather measurements from different sources and to specify as accurately as possible the wind characteristics on the sea, off Norway. But the amount of data provided by this campaign is small (22,000 individual observations not totally independent) and it is not representative of the whole range of the wind speed and direction in the atmosphere: it only covers the wind speed range of 1-21 m/s.

There exist other data bases gathering wind vectors: Rufenach (1995) made use of 75,000 collocated buoy measurements to estimate upwind-crosswind σ^o differences.

Moreover, this data is not free from noise. On the one hand, there exists measurement errors: they are due to the imperfection of the instruments. On the other hand there are also interpretation errors: they are due to differing scales to which the observations relate. Indeed, in order to make the RENE-91 measures relevant for a comparison with satellite measurements of σ^o, they must be averaged or interpolated in time or in space to match the characteristics of the satellite footprint.

Finally, they must be adjusted to a reference height of 10 m. For instance, buoys measure winds at a height of 3 or 4 m and aircraft measure the wind at their flying height. All these calculations add uncertainty to the measured values.

As ground measurements do not provide enough information to train a model on a wide range of speeds, they were not used in this study.

2.1.2 Numerical models

Presentation

Numerical models provide an alternative source of data which is used for this study. Unlike ground measurements, the European Centre for Medium-Range Weather Forecast (ECMWF) produces a complete wind field over the globe, with all speeds and directions represented to at least some extent. In this study, the data files which were utilised to train the neural networks come from the Centre ERS d'Archivage et de Traitement (CERSAT[1]). CERSAT makes use of ECMWF wind fields interpolated to the position and time of the satellite measurements.

Data selection and filtering

Filtering at CERSAT CERSAT operates a rather stringent quality control on the data gathered by the ERS-1 satellite (Maroni *et al.*, 1995). CERSAT generates its own wind field products (called WNF) with associated σ^o values. Associating σ^o to ECMWF winds gives rise to errors as detailed in next section. CERSAT discards some σ^o measurements.

[1]Information about CERSAT can be found at `http://www.ifremer.fr/cersat/`

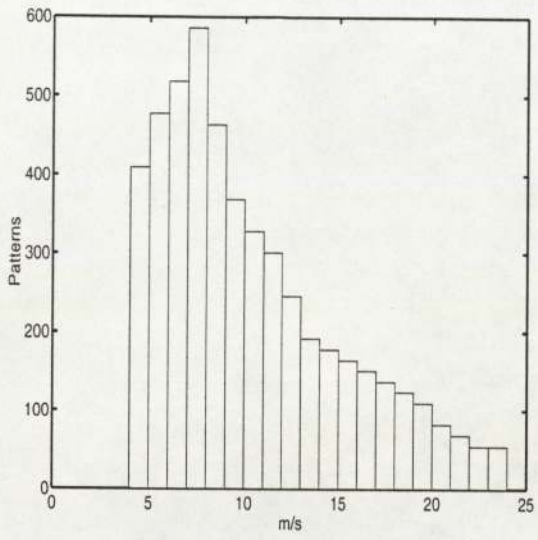
Reasons for not validating σ^o measurements include: sea-ice data (a dynamic sea-ice mask is used), low winds ($< 3\ m/s$) for which the σ^o measurements are too noisy, points contaminated by high rain or snow rates. The three latter reasons are revealed by a Maximum Likelihood Estimator (MLE) being out of range, *i.e.* the σ^o triplet lying too far away from the manifold defined by CMOD-IFR2 forward model in σ^o space. According to the results in this thesis, CMOD-IFR2 is not a very good model for an unconditional probability distribution of the measurements in σ^o space.

Further selection A further selection of the data used in this study was carried out. Only cells from the 10 odd tracks were selected. This can be justified by the fact that the odd and the even cells completely overlap. The distribution of the incidence angle θ was smoothed. A smooth distribution of relative wind directions was also ensured. Finally, the speed distribution of the data sets was chosen to represent both the natural atmospheric distribution and our desire to get a good prediction of high wind speeds. This was intended to avoid the use of weighted cost functions (Cornford and Nabney, 1998a), and it was made possible thanks to a sufficient amount of data. So the distribution of wind speeds for the training data set is the equal sum of a uniform distribution and the atmospheric distribution (see Figure 2.2), in the range 4 - 24 m/s. To evaluate the error of the model, we used a test data set with a speed distribution corresponding to the atmospheric distribution (Figure 2.1).

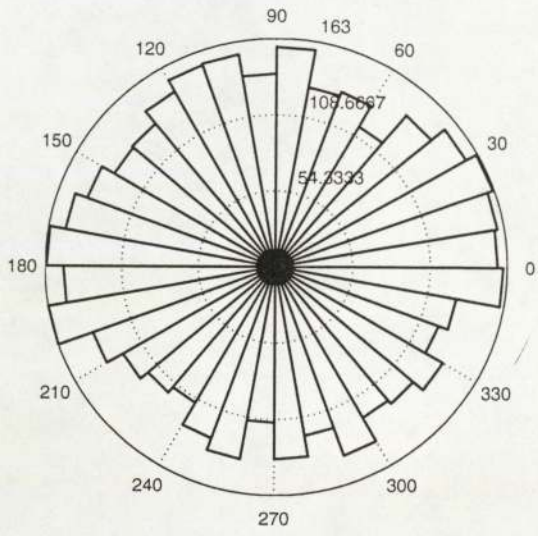
It is shown in Section 5.2 that using a test and validation sets to compute test and validation errors is actually not justified. Important issues in the process of data selection are also discussed in section 6.1.

Error analysis

The error of ECMWF predicted winds against true (unknown) winds is thought to have a Gaussian distribution in each Cartesian component of the wind, with the same standard deviation. As resolution of the wind fields for this model (250 by 250 km) is lower than the resolution of the cells on the footprint of the satellite (50 by 50 km), spatial interpolation is required. This gives rise to a second source of error. Moreover, temporal adjustment is also required. Finally, the reference height of the ECMWF model we use provides winds



(a) Distribution of the wind speed (atmospheric distribution)



(b) Distribution of the wind relative direction

Figure 2.1: Characteristics of the wind vectors in the test set (file `dirs_tst`, 5,000 patterns. All incidence angles are equally represented and the directions are rather smoothly distributed.

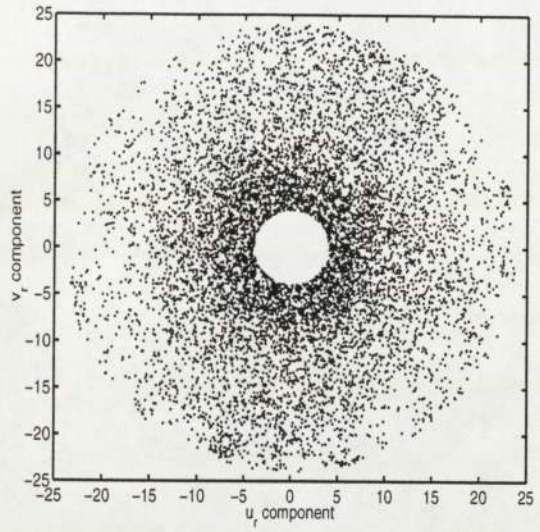
at a height of 50 m. As for ground measurements, these must be adjusted to the reference height of 10 m (this is done for us by ECMWF), giving rise to even more uncertainty.

To summarise, the level of errors in WNF winds is not well known. However, the error distribution seems best described as a Gaussian distribution in (u, v) space. Stoffelen and Anderson (1997b) found that the standard deviation of the error in both u and v wind components is 1.5 m/s. In $(\|\boldsymbol{u}\|,\vartheta)$ space, the errors have complicated skewed distributions. Some aspects of this topic will be discussed in greater detail in Chapter 5. These errors are not taken into consideration at the beginning of the work presented in this thesis.

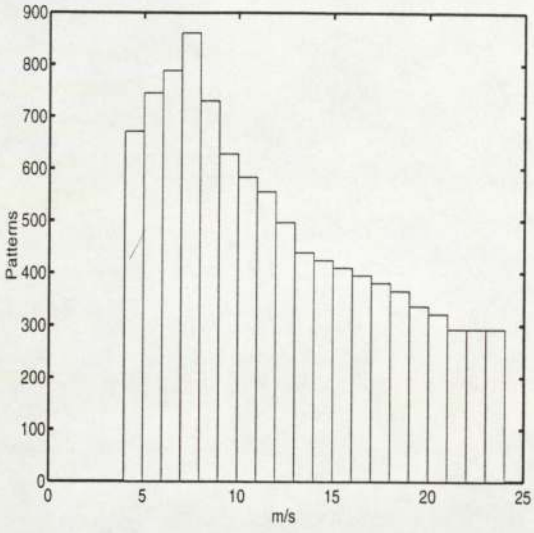
Concerning the $\boldsymbol{\sigma^o}$ values, Stoffelen and Anderson (1997a) showed that the errors are proportional in linear space. Therefore $\boldsymbol{\sigma^o}$ values are measured in logarithmic space in all this study:

$$\sigma^0_{dB} = 10 \log_{10} \sigma^0_{lin} \tag{2.1}$$

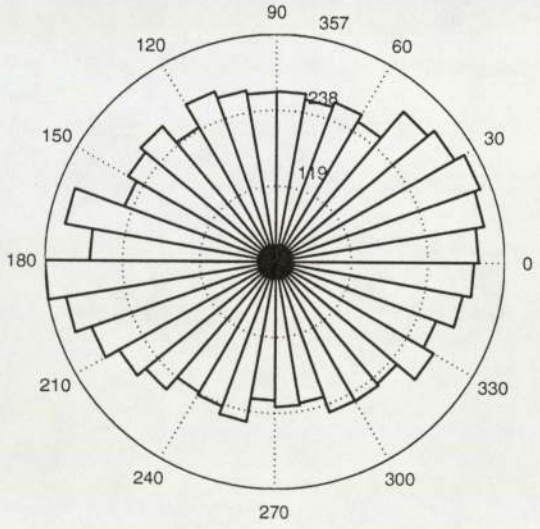
Errors are additive in log space, and their typical spreading from the theoretical surface



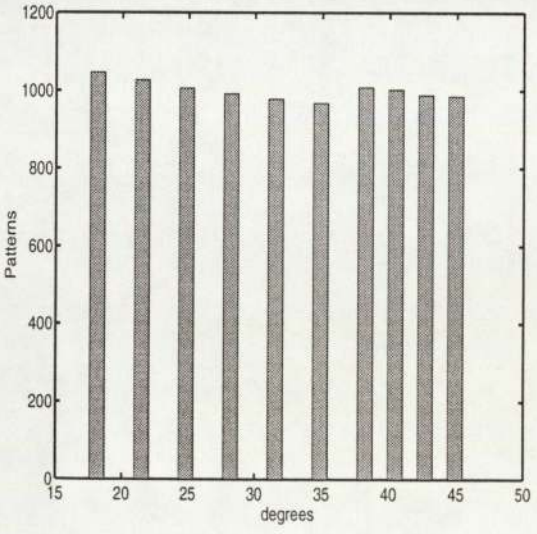
(a) The wind vectors in (u, v) space



(b) Distribution of the wind speed: atmospheric + uniform



(c) The distribution of the wind direction is rather smooth



(d) Repartition of wind patterns between the 19 tracks. Only the 10 odd tracks were sampled from.

Figure 2.2: Characteristics of the wind vectors in the training set (file `dirs_trn`, 10,000 patterns)

on which they lie is 0.2 dB (Stoffelen and Anderson, 1997a).

2.2 Data file format

The fields of the data files used to train neural networks are shown on Table 2.1. Different files were used at the end of this study.

field	symbol	description	units
1	σ_f^o	fore beam backscatter	dB
2	σ_m^o	mid beam backscatter	
3	σ_a^o	aft beam backscatter	
4	θ	incidence angle between local vertical and satellite *mid* beam	degrees
5	χ	azimuth: angle between North and mid beam	degrees
6	$\|\boldsymbol{u}\|$	wind speed	m/s
7	mdir	meteo wind direction	degrees

Table 2.1: Fields of the data files

Here is a short description of each field: θ can take 19 values (only 11 values appear) corresponding to the 19 tracks on the swath, from 17.9° to 45.5° (this is mid beam). In order to run composite models such as CMOD4, we used a lookup table to obtain the incidence angles of fore and aft beams.

The azimuth angle χ is bimodal: it oscillates around the two values 79 or 281° depending on whether the satellite is on an ascending or descending path.

More symbol conventions are shown in Appendix A.

2.3 Modelling the expectation using a RBF network

As a first approach, only the conditional average of the backscatter from the ocean was modelled: $E[\boldsymbol{\sigma^o} \mid u, v, \theta]$. This was done using either a Radial Basis Function (RBF) or a Multi Layer Perceptron (MLP) network. These neural networks are described in (Bishop, 1995). They were trained as complete models (Figure 2.3) so as to prepare for modelling a complete probabilistic distribution of $\boldsymbol{\sigma^o}$. As far as Root Mean Square (RMS) errors and biases are concerned, both give similar results, the RBF doing slightly better though.

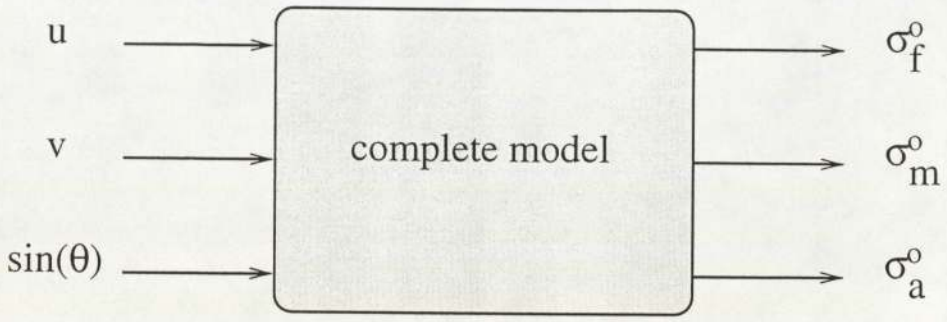


Figure 2.3: The models presented in this section have 3 outputs. They take the Cartesian wind components and the sine of the incidence angle as inputs.

The RMS error is defined for each beam as:

$$E_{\mathrm{RMS}} = \frac{1}{N}\sum_{1}^{N}(\sigma^o_{i,s} - \sigma^o_{i,o})^2 \tag{2.2}$$

where $i \in \{1,2,3\}$ indicates beam and $\sigma^o_{i,s}$ and $\sigma^o_{i,o}$ are the N simulated and observed measurements. The bias is $\sum_1^N(\sigma^o_{i,s} - \sigma^o_{i,o})\,/N$.

2.3.1 Pre/post-processing

All the data sets are pre-processed in the same way to train the networks. Training input as well as output data is normalised so it has zero mean and unit variance (for each field independently) as suggested by Bishop (1995). This ensures that all input and output variables are of order unity, and we expect the network weights to be in the same range of values. The sine of the incidence angle was also used instead of the angle itself.

In practice, once the network is trained, input data to the network has to be translated and rescaled with the same coefficients as the training set. The output of the neural network then has to be rescaled using the inverse transformation.

2.3.2 Network architecture

The RBF network has a 3-dimensional input space and a 3-dimensional output space. The hidden unit activation functions are Gaussian. Several values for the number of hidden units are compared. There is still no over-fitting with up to 100 hidden units (both

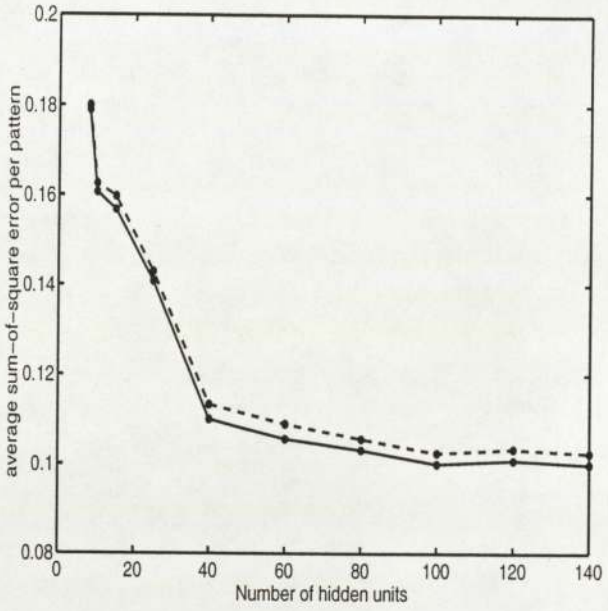


Figure 2.4: Plot of the sum-of-square error as a function of the number of hidden units of the RBF network.

training and validation sum-of-squares errors go on decreasing when increasing the number of hidden units). The lowest test error for this network is obtained with the surprisingly large number of 100 hidden units, as shown in Figure 2.4.

2.3.3 Results

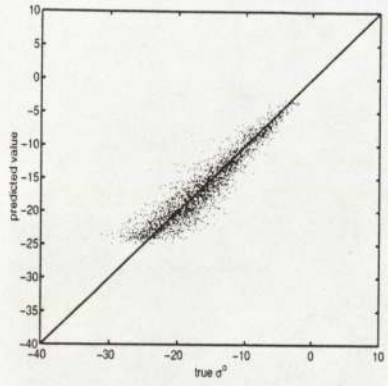
Here are are some basic statistics about this model. For the test set, the covariance matrix of the residuals of $(\sigma_f^o, \sigma_m^o, \sigma_a^o)$ of the network is:

$$\Sigma_{\mathbf{r}} = \begin{pmatrix} 0.0929 & 0.0508 & 0.0474 \\ 0.0508 & 0.0456 & 0.0456 \\ 0.0474 & 0.0456 & 0.0767 \end{pmatrix}$$

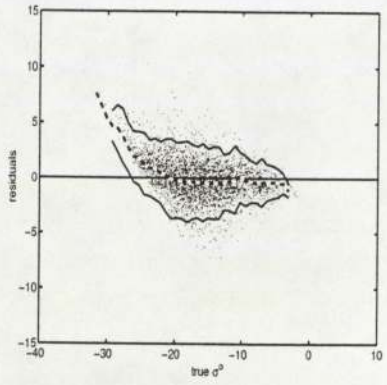
And their bias is almost zero:

$$\mathbf{b_r} = \begin{pmatrix} -0.0018 & -0.0009 & -0.0029 \end{pmatrix}$$

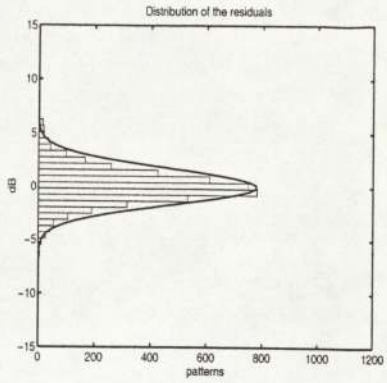
Some indicators of the performances of this neural network are displayed on Figure 2.5. The residuals are plotted in the middle column. The thick dashed line represents



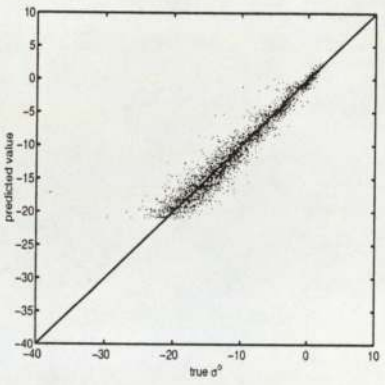
(a) Computed value of σ^o against true value



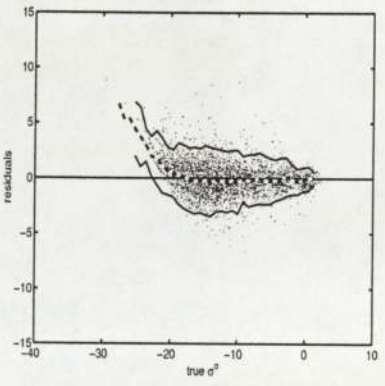
(b) Residuals against true value (fore beam)



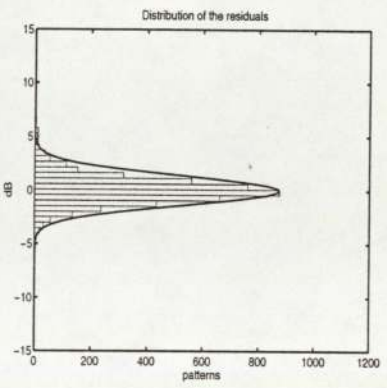
(c) Distribution of the residuals



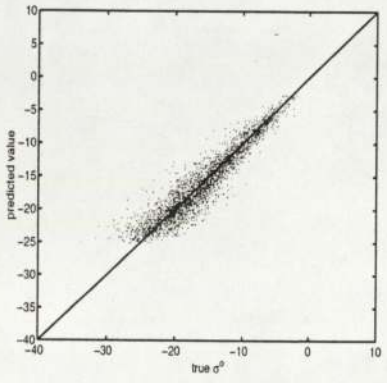
(d) Computed value of σ^o against true value



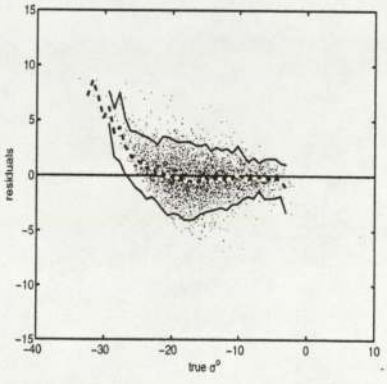
(e) Residuals against true value (mid beam)



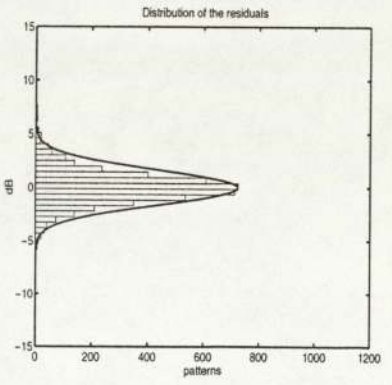
(f) Distribution of the residuals



(g) Computed value of σ^o against true value



(h) Residuals against true value (aft beam)



(i) Distribution of the residuals

Figure 2.5: Performances of the RBF network with 100 hidden units. Figures on lines 1, 2 and 3 represent fore, mid and aft beams respectively.

the moving average of the residuals. It appears clearly that the low values of σ^o are overestimated, by up to 5 dB for σ^o measured values below -30 dB. Fortunately only a small proportion of the patterns are concerned. On the contrary, higher values of σ^o are slightly underestimated. Finally, the bias is almost zero: on the third column of Figure 2.5, the histograms show the distribution of the residuals, and a centred Gaussian curve on top of each of them. Each histogram fits the Gaussian very well.

At this stage of the work, this was considered a good omen, as representing $P(\sigma^o \mid u, v, \theta)$ using a Gaussian model would be achieved with a good approximation. The standard deviation of the residuals was also computed over small intervals, in the same way as for the moving average. The two thin lines on each plot of the middle column represent the mean $\pm$ twice the standard deviation. This standard deviation is not constant: it is lower for extreme values of σ^o. Here again, this is validating the choice for a probabilistic model using input-dependent covariance matrices.

2.4 Modelling the expectation using a MLP network

2.4.1 Complete model

Several Multi Layer Perceptrons (MLP) were also trained to perform this task. Two different methods of regularization were applied: early stopping and weight decay. Early stopping showed there was no over-fitting: after 600 training cycles using the Scaled Conjugate Gradient algorithm and with a number of hidden units as large as 50, the test error was still slowly decreasing.

Finally, the results were found to be close to those obtained using RBF networks. The covariance matrix of the residuals is not diagonal either, showing that the residuals are correlated:

$$\Sigma = \begin{pmatrix} 0.1160 & 0.0646 & 0.0550 \\ 0.0646 & 0.0698 & 0.0582 \\ 0.0550 & 0.0582 & 0.1029 \end{pmatrix}$$

Moreover, the residuals also had a normal distribution (as in Figure 2.5, not shown). This suggested the use of a multivariate density network to model $P(\sigma^o \mid u, v, \theta)$.

2.4.2 Composite model

A composite forward model is developed using an MLP. The inputs are the same, but there's only one output. The model is run three times to get the σ^o triplet. Several levels of complexity are tested as above. The results are not different from the complete models.

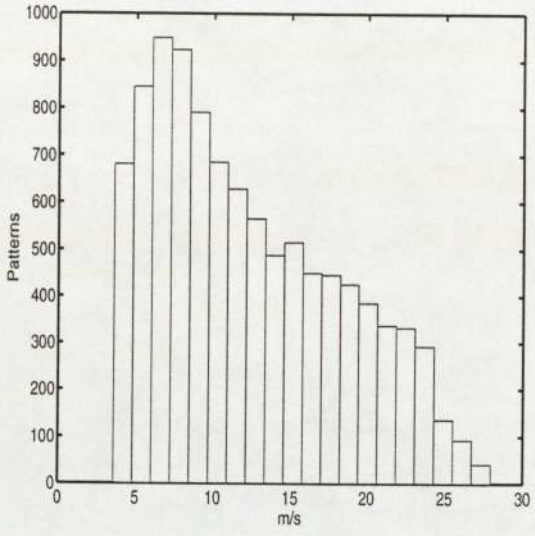


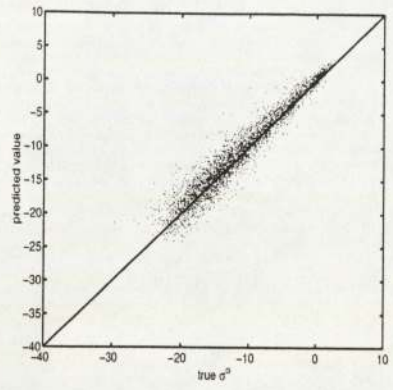
Figure 2.6: Speed distribution in the datafile `so_trn`: several composite models were trained on this file.

It should be noted that training a composite model requires a different file format: the three fields σ^o_f, σ^o_m, σ^o_a are replaced by a single field σ^o, in which the values are taken from σ^o_f, σ^o_m, σ^o_a with equal probabilities. In such a file, the azimuth angle χ takes 6 different values instead of 2. The same file `so_trn` was used to train the hybrid model presented in section 3.2. Its speed distribution is shown in Figure 2.6. There are 10,000 patterns.

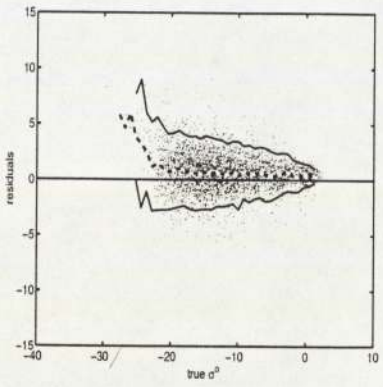
2.5 Comparison with other models

CMOD4 and CMOD-IFR2 are known to over-estimate σ^o values at high wind speeds. Indeed they show a strong positive bias when their output is computed from our ECMWF wind samples and compared with collocated σ^o values.

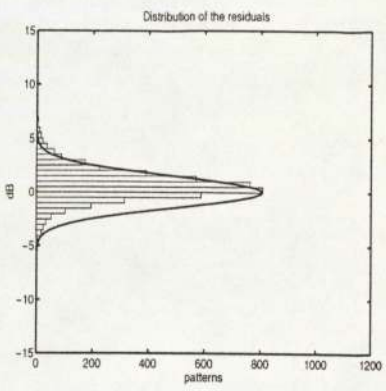
As stated in (Mejia *et al.*, 1998), NN-GMF exhibits a lower bias than the two models above. Tables 2.2 and 2.3 give the values of the bias and RMS error for each model and



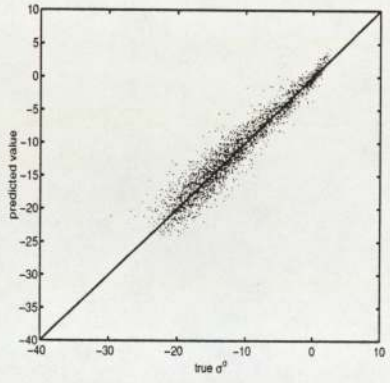
(a) Computed value of σ^o against true value



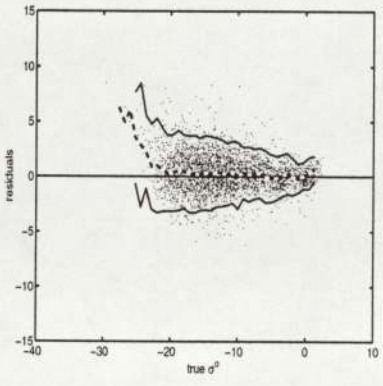
(b) Residuals against true value



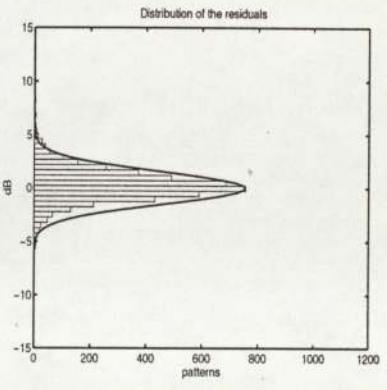
(c) Distribution of the residuals



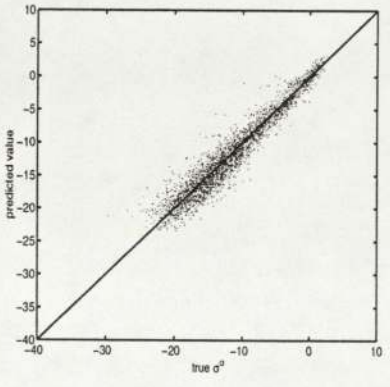
(d) Computed value of σ^o against true value



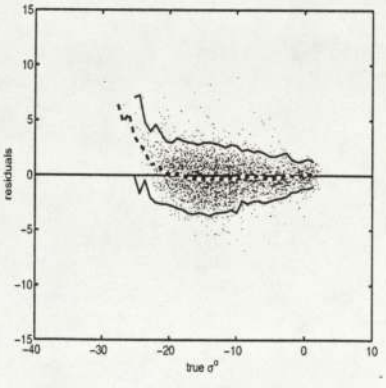
(e) Residuals against true value



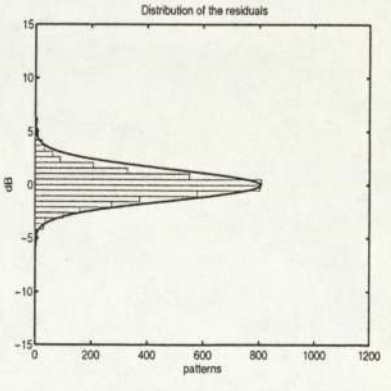
(f) Distribution of the residuals



(g) Computed value of σ^o against true value



(h) Residuals against true value



(i) Distribution of the residuals

Figure 2.7: Residuals of three reference models for mid beam measurements. CMOD4, CMOD-IFR2 and NN-GMF are presented on lines 1, 2 and 3 respectively. All incidence angles are present. Test file is `dirs_tst` (4-24 m/s).

each beam. The residuals of the three reference models are also plotted in Figure 2.7.

Further assessment of the accuracy of these models (and also VIERS-1) is given in Chapter 4.

	CMOD4		CMOD IFREMER		NN-GMF	
	bias	RMS error	bias	RMS error	bias	RMS error
Fore beam	0.7660	1.9412	0.3528	1.8575	-0.1966	1.7338
Mid beam	0.6284	1.6042	0.2646	1.5419	-0.1021	1.4614
Aft beam	0.7369	1.8543	0.3240	1.7480	-0.2053	1.6957

Table 2.2: Bias and RMS error (both in dB) of three reference forward models tested with ECMWF input winds in the range 4 - 18 m/s (selected from file `dirs_tst`).

	RBF 100 h.u.		MLP 60 h.u.		composite MLP 20 h.u.	
	bias	RMS error	bias	RMS error	bias	RMS error
Fore beam	-0.0069	1.6975	-0.0069	1.6663	0.0280	1.6838
Mid beam	-0.0018	1.3711	-0.0029	1.3564	-0.0854	1.3805
Aft beam	-0.0106	1.6196	-0.0069	1.6033	0.0081	1.6270

Table 2.3: As above for an RBF with 100 hidden units, an MLP with 40 hidden units and a single output MLP with 20 hidden units. All speeds were used in the test file (up to 24 m/s). Neural networks outperform reference models but these values are a poor indicator of quality of models.

Chapter 3

More advanced models

The models which are presented in this chapter are "more advanced": their architecture is still based on MLP neural networks but the outputs are subject to further treatment compared to conventional MLPs.

3.1 Multivariate Density Networks

The Multivariate Density Network (MVDN) is a probabilistic model. As a first approach before its development, a mixture density network (MDN) was trained for the same task.

3.1.1 First attempt with mixture density networks

Introduction to Mixture Density Networks

In order to model the conditional probability density $P(\sigma^o \mid u, v, \theta)$, an MDN was trained. Unfortunately, this model didn't give any result because it was too computationally expensive[1]. This type of model combines a conventional neural network whose outputs determine the parameters in a mixture density model. It can in principle represent arbitrary conditional probability distributions. The probability density of the target data, t,

[1]Now there exists a new faster implementation for this model (Evans, 1998), but it was not available at that time.

is represented as a linear combination of kernel functions in the form:

$$p(\boldsymbol{t} \mid \boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{x}) \phi_i(\boldsymbol{t} \mid \boldsymbol{x})$$

where $\phi_i(\boldsymbol{t}|\boldsymbol{x})$ represents the conditional density of the target vector $(\boldsymbol{t})$ for the i^{th} kernel.

This type of neural network is extensively described in (Bishop, 1994).

Limitations

The structure for MDNs is implemented in the NETLAB[2] toolbox for MATLAB. This implementation uses Gaussian kernels and it only supports spherical covariance matrices. MDNs are particularly useful to represent multi-modal distributions. Although the model we want to build is unimodal, it would have still been interesting to train an MDN with several kernels for our problem. Indeed, in a 3D space for instance, two overlapping spherical kernels can provide an approximate representation of an ellipsoid. However, after a few trials it was estimated that training one single model would take at least 50 days of computation. It was not worth improving the code as it was not the most appropriate model anyway.

3.1.2 Presentation of the Multivariate Density Network

Like an MDN, an MVDN provides a way of modelling conditional probability distributions. (Williams, 1996) gives a complete description of this model. The basic principles are recalled here for convenience. The conditional distribution of the n-dimensional quantity $\boldsymbol{y}$ given $\boldsymbol{x}$ is assumed to be described by the multivariate density

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}) \right\}$$

where $\boldsymbol{\mu}(\boldsymbol{x})$ is the vector of conditional means and $\boldsymbol{\Sigma}(\boldsymbol{x})$ is the conditional covariance matrix.

These parameters are determined by the outputs of the network: Obtaining the mean vector $\boldsymbol{\mu}$ presents no problem. Its n components are given by n network outputs without

[2]NETLAB is available from `http://www.ncrg.aston.ac.uk/netlab/`

any further treatment. Obtaining the covariance matrix is less obvious. As $\mathbf{\Sigma}$ is symmetric, there are only $n(n+1)/2$ independent entries. $\mathbf{\Sigma}$ is also positive definite so its inverse can be written as follows, using the Cholesky factorisation, with $n = 4$ for example:

$$\mathbf{\Sigma}^{-1} = A^T A = \begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{12} & \alpha_{22} & 0 & 0 \\ \alpha_{13} & \alpha_{23} & \alpha_{33} & 0 \\ \alpha_{14} & \alpha_{24} & \alpha_{34} & \alpha_{44} \end{pmatrix} \begin{pmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ 0 & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ 0 & 0 & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & \alpha_{44} \end{pmatrix} \tag{3.1}$$

This decomposition exhibits the $n(n+1)/2$ independent entries. The diagonal terms are positive. Therefore they are related to n network outputs by the exponential function. Off-diagonal entries are unconstrained so they are taken directly from $n(n-1)/2$ other outputs. Using the same notation as in (Williams, 1996):

$$\begin{aligned} \alpha_{ii} &= \exp(z_i^{\pi}) \qquad && i = 1 \ldots n \\ \alpha_{ij} &= z_{ij}^{\alpha} && i = 1 \ldots n-1, \quad j = 2 \ldots n, \quad i < j. \end{aligned}$$

There are $n(n+3)/2$ entries overall. This is also the number of required outputs for the neural network.

The negative log-likelihood of a set of N observations can be written as:

$$E = \sum_{p=1}^{N} [\frac{1}{2} \log |\mathbf{\Sigma}_p| + \frac{1}{2} (\boldsymbol{y}_p - \boldsymbol{\mu}_p)^T \mathbf{\Sigma}_p^{-1} (\boldsymbol{y}_p - \boldsymbol{\mu}_p)]$$

where p is the index for each pattern and the normalisation constant has been dropped. The derivatives are computed analytically and the network is trained using a non-linear optimisation algorithm. More details and results can be found in (Williams, 1996).

3.1.3 Implementation of the Multivariate Density Network

In order to keep some compatibility with NETLAB, the same structure as in the implementation for the mixture density network was kept. Thus the programs were called `mvdn` `mvdninit` `mvdnfwd` `mvdngrad` `mvdnerr` `mvdnpak` and `mvdnunpak`.

`mvdn`	creates the network part of the model (an MLP) and the structure for the covariance matrix.
`mvdninit`	initialises the weights of the network. The target data is used to set the output biases of the network: the mean of the target and the Cholesky decomposition of the covariance matrix of the residuals is computed. The weights which correspond to the output biases are then initialised to yield these values. The other weights are initialised randomly with a Gaussian prior.
`mvdnfwd`	forward propagates the inputs through the model. The output is an array of structures containing the mean vector and the covariance matrix for each input pattern.
`mvdnerr`	computes the error of the model for a set of inputs and targets.
`mvdngrd`	computes the error gradient of the model.
`mvdnpak/unpak`	packs the weights of the network into a vector: this is required to use the scaled conjugate gradient optimisation.

An extra program called `mlp2alpha` performs the transformation from the MLP vector of outputs to the upper triangular matrix A (Equation 3.1). The elements of A were numbered as follows (this would be in dimension 4):

$$\begin{pmatrix} 1 & 5 & 6 & 7 \\ 0 & 2 & 8 & 9 \\ 0 & 0 & 3 & 10 \\ 0 & 0 & 0 & 4 \end{pmatrix}$$

3.1.4 Computational efficiency

Because of the presence of structures (containing both the mean vector and the covariance matrix for each pattern) in the model, it was not directly possible to vectorise the code.

MATLAB is very poor at using loops so running times were going to be even longer than for the MDN. Hence the most computationaly intensive parts of the code (three loops over the 10,000 patterns) were re-written in C, with an interface with MATLAB. The time of execution of the whole code was reduced by a factor of 88. It still took about ten hours to train a model for 1,000 iterations of the scaled conjugate gradient (SCG) algorithm but this is quite acceptable.

3.1.5 Training the Multivariate Density Network

For this model, the pre/post-processing are exactly the same as for networks where only the expectation is modelled (section 2.3.1). It is trained using the file `dirs_trn`. However, some care has to be taken when un-normalising the outputs given by the model. Indeed, if $\widetilde{\Sigma}$ denotes the covariance matrix given by the multivariate density network immediately before post-processing, then the elements of the post-processed covariance matrix Σ are given by:

$$\Sigma_{i,j} = \widetilde{\Sigma}_{i,j} * M_{i,j} \qquad i,j \in \{1,2,3\} \tag{3.2}$$

where M is the outer product:

$$M = \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix} * \begin{pmatrix} \sigma_1 & \sigma_2 & \sigma_3 \end{pmatrix} \tag{3.3}$$

where $(\sigma_1 \ \sigma_2 \ \sigma_3)$ is the vector of the three standard deviations of $(\sigma_f^o, \sigma_m^o, \sigma_a^o)$ in the training set, which are used for rescaling the targets (pre-processing).

Various MVDNs with different complexities were trained using the SCG algorithm, with the file `dirs_trn` (see section 2.1 for a description). All of them were trained for at least 400 iterations and some of them for up to 1,000 iterations. Although the error level seemed not to decrease significantly during these extra iterations, graphical representations exhibited a real evolution (this was discovered later).

Studying the residuals always gives similar results: the bias is almost zero and the

distribution of the residuals (measured $\boldsymbol{\sigma}^o$ minus computed most probable $\boldsymbol{\sigma}^o$) was very similar to the distribution found for networks modelling the expectation $E[\boldsymbol{\sigma}^o \mid u, v, \theta]$, whatever the complexity. For instance, the biases for the MVDN with 10 hidden units were as low as (0.070 0.080 0.066) after only 400 iterations. The test errors were nearly identical for all MVDNs.

At this stage it was necessary to find a better way of comparing these models. Graphical means are presented in the next chapter. The cross-section and the vertical section of the cone in $\boldsymbol{\sigma}^o$ space were used to assess the quality of the various MVDNs.

3.2 Hybrid neural network

As will be shown in Chapter 4, conventional neural networks that are trained on the (u, v) to $\boldsymbol{\sigma}^o$ mapping exhibit unwanted asymmetries. This section presents a more complicated model which corrects this fault. The assumption of a symmetric mapping with respect to ϑ is based on the symmetric geometry of the scatterometer measurement.

3.2.1 Symmetry in CMOD4 and CMOD-IFR2

A close study of CMOD4's definition shows that the dependency of the outputs of the model on the wind direction is fixed:

$$\sigma^0_{\text{lin}} = B_0(1 + B_1 \cos\vartheta + B_3 \tanh B_2 \cos 2\vartheta)^{1.6} \tag{3.4}$$

In the expression above, the parameters B_0, B_1 and B_2 do not depend on ϑ, so the model is constrained to be symmetric with respect to ϑ. These three parameters are complicated functions of $\|\boldsymbol{u}\|$ and θ. These functions include hyperbolic dependencies, Legendre polynomials and various thresholds. They introduce further constraints on the dependency of the parameters B_0, B_1 and B_3 on $\|\boldsymbol{u}\|$ and θ.

The basic idea of the hybrid neural network introduced here is to copy the CMOD4 functional form and to determine its parameters using a neural network instead of the functions described above.

Here is a reminder of the functional form of CMOD-IFR2:

$$\sigma^0_{\text{lin}} = B_0(1 + B_1 \cos\vartheta + B_3 \tanh B_2 \cos 2\vartheta) \tag{3.5}$$

This is a truncated Fourier series where only the first two symmetric components were retained. (Stoffelen and Anderson, 1997b) introduces the transformation $z = (\sigma^o)^{0.625}$ (hence the $1.6 = 1/0.625$ in Equation 3.4) in order to compensate for the absence of higher harmonics $\cos 3\vartheta$, $\cos 4\vartheta$. This value 0.625 was determined by empirical means and it may not reflect the varying relative importance of the harmonics. Therefore this becomes a tunable parameter in the hybrid neural network model. This model is called 'nn2cmod' hereafter. A sketch of this model is shown in Figure 3.1.

3.2.2 New functional form

A natural functional form for nn2cmod could have been:

$$\sigma^0_{\text{lin}} = B_0(1 + B_1 \cos\vartheta + B_2 \cos 2\vartheta)^p \tag{3.6}$$

But in the expression above the term $(1 + B_1 \cos\vartheta + B_2 \cos 2\vartheta)$ has to be strictly positive so the expression is defined for all p. This is achieved with the following transformation (the variables B_0, B_1 and B_2 are not consistent from one equation to the other):

$$\sigma^0_{\text{lin}} = B_0(1 + 0.1 \tanh(B_1) \cos\vartheta + 0.8 \tanh(B_2) \cos 2\vartheta)^p \tag{3.7}$$

In Equation 3.7, the values 0.1 and 0.8 are just scaling parameters. They were chosen so as to sum to a value less than 1. Their relative values have little importance as the network output weights will incorporate them. Finally, σ^o can be written in log space:

$$\sigma^0_{\text{log}} = \underbrace{\frac{10}{\ln(10)}}_{=L} \Big(b_0 + p \ln \underbrace{\big(1 + 0.1 \tanh(b_1) \cos(\vartheta) + 0.8 \tanh(b_2) \cos(2\vartheta)\big)}_{=K} \Big) \tag{3.8}$$

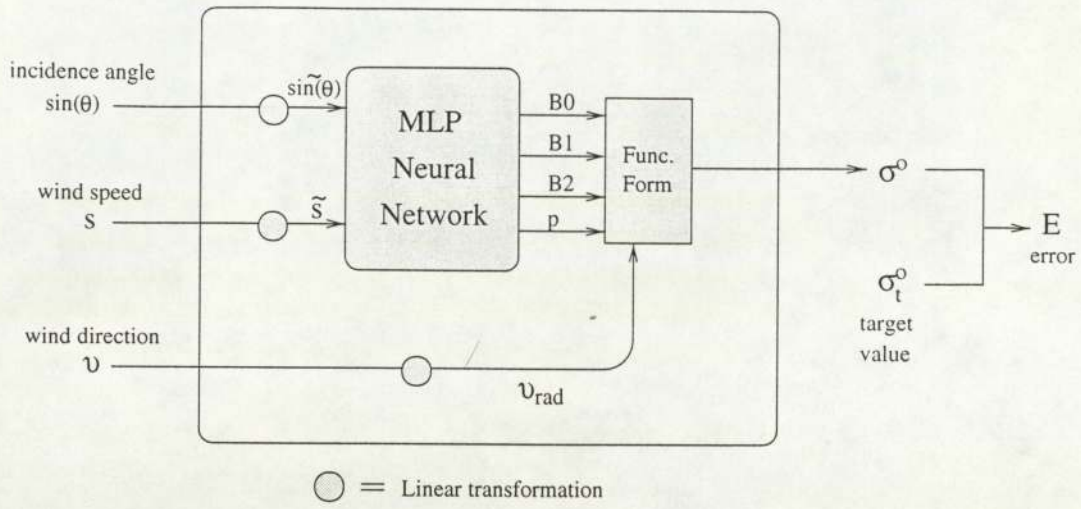


Figure 3.1: Sketch of the hybrid model. The wind direction is no longer an input to the neural network. Variables denoted with a tilde are normalised variables (set to zero mean and unit variance by a linear transformation).

3.2.3 Error function and its gradient

Assuming Gaussian errors in σ^o space, the error of the model is given by the following sum-of-squares error, where the normalising constants are removed:

$$E = \sum_{n=1}^{N} E_n \tag{3.9}$$
$$= \frac{1}{2\sigma_{\sigma^o}^2} \sum_{n=1}^{N} (\sigma_n^o - \sigma_{n,t}^o)^2$$

where N is the number of scatterometer measurements in the dataset. To train this model, the expression of the derivatives of the error with respect to the weights is necessary.

Using the notation introduced in Equation 3.8, the derivatives of the error function with respect to the outputs of the neural network can be written as follows. For clarity,

the derivatives are calculated for each pattern but the subscripts n are removed.

$$\begin{aligned}
\frac{\partial E}{\partial B_0} &= L\,\frac{(\sigma^o - \sigma_t^o)}{\sigma_{\sigma^o}^2} \\
\frac{\partial E}{\partial B_1} &= \frac{0.1\,L\,p\cos\vartheta}{K\,\cosh^2 B_1}\,\frac{(\sigma^o - \sigma_t^o)}{\sigma_{\sigma^o}^2} \\
\frac{\partial E}{\partial B_2} &= \frac{0.8\,L\,p\cos 2\vartheta}{K\,\cosh^2 B_2}\,\frac{(\sigma^o - \sigma_t^o)}{\sigma_{\sigma^o}^2} \\
\frac{\partial E}{\partial p} &= L\,\ln(K)\,\frac{(\sigma^o - \sigma_t^o)}{\sigma_{\sigma^o}^2}
\end{aligned} \tag{3.10}$$

The derivatives of the error function with respect to the weights of the network are then obtained using back-propagation (Bishop, 1995).

3.2.4 Training the model

No particular theoretical problems arose for the implementation of the nn2cmod model. The structure is similar to the one used for the Multivariate Density Network. The files `nn2cmoderr` `nn2cmodfwd` and `nn2cmodgrad` played the same roles as for the MVDN (see table page 34). The normalisation of the data is different from the other models as the input ϑ is not connected to the network and σ^o is no longer an output. The ranges of values of the four outputs of the MLP are not normalised but they are reasonably close to zero, and their magnitude is small.

The model was trained for 1,000 iterations[3] using the scaled conjugate gradient algorithm. The training data file had 10,000 patterns (the file is briefly described in Section 2.4.2). The network was trained for different numbers of hidden units: 10, 20, 30 and 40. The output biases of the network were initialised using information from CMOD4: CMOD4 was run on the training set and the average values obtained for B_0, B_1 and B_2 were computed. Some necessary transformation was carried out in order to yield the definitions of B_0, B_1 and B_2 in nn2cmod. Eventually the output biases were initialised to the three following values: $(-1.8, 0.07, 0.3)$ and 1.6 for the fourth output bias associated with p.

[3]The error had reached a stable level whatever the complexity

3.2.5 Results

The consistency of these four networks was checked by comparing their respective four outputs. Figures 3.2 and 3.3 show these outputs as functions of speed and mid beam incidence angles. They had the same shape for the four networks (note that the presented plots are the most different cases). In early trials where the output biases had not been set, the outputs of two different networks (20 and 15 hidden units) finished in a very different state where p took values around -4 although the cone surface was fitting the points rather correctly in σ^o space (see next chapter).

This model has several advantages. As will be seen in the next chapter, its behaviour at high wind speeds is good although the data is sparse (Figure 4.7). It means the risk of over-fitting the data is smaller. This can be explained by two reasons. First the model is symmetric so 50% less data is required for the same accuracy. Second, the outputs of the MLP have a smoother dependency on the inputs so it is likely that the variation of these will also be smooth where the training data is sparse or even absent. One drawback is the assumption of symmetry. There's no proof of its validity.

In chapter 5, this model is reused as it exhibits the best features so far.

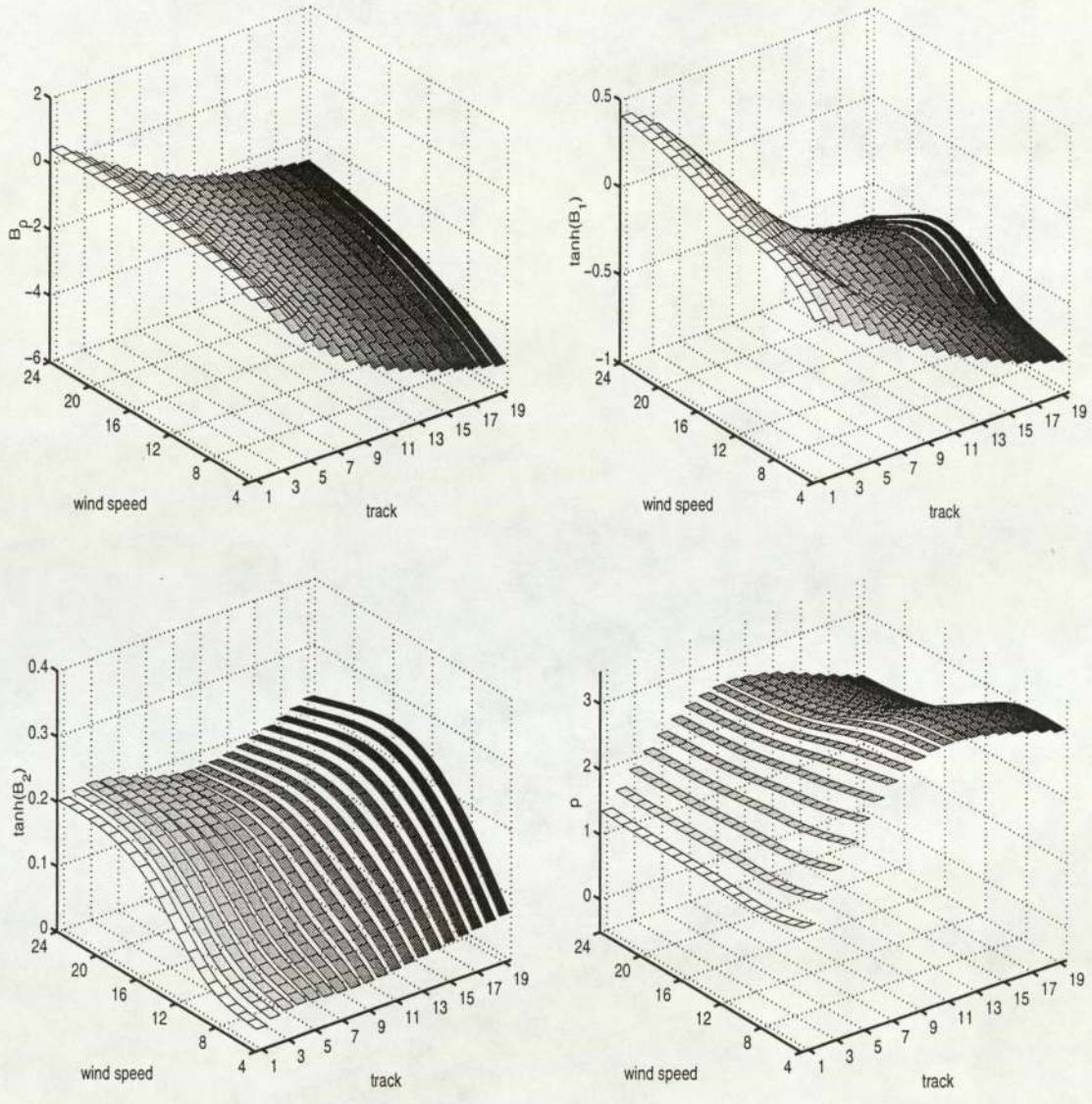


Figure 3.2: Outputs of the model NN2CMOD with 10 hidden units (one plot per output). They are consistent with the values obtained with 20, 30 (not shown) and 40 hidden units (see next page). The range of values obtained for p is quite comparable to the fixed value (1.6) of CMOD4.

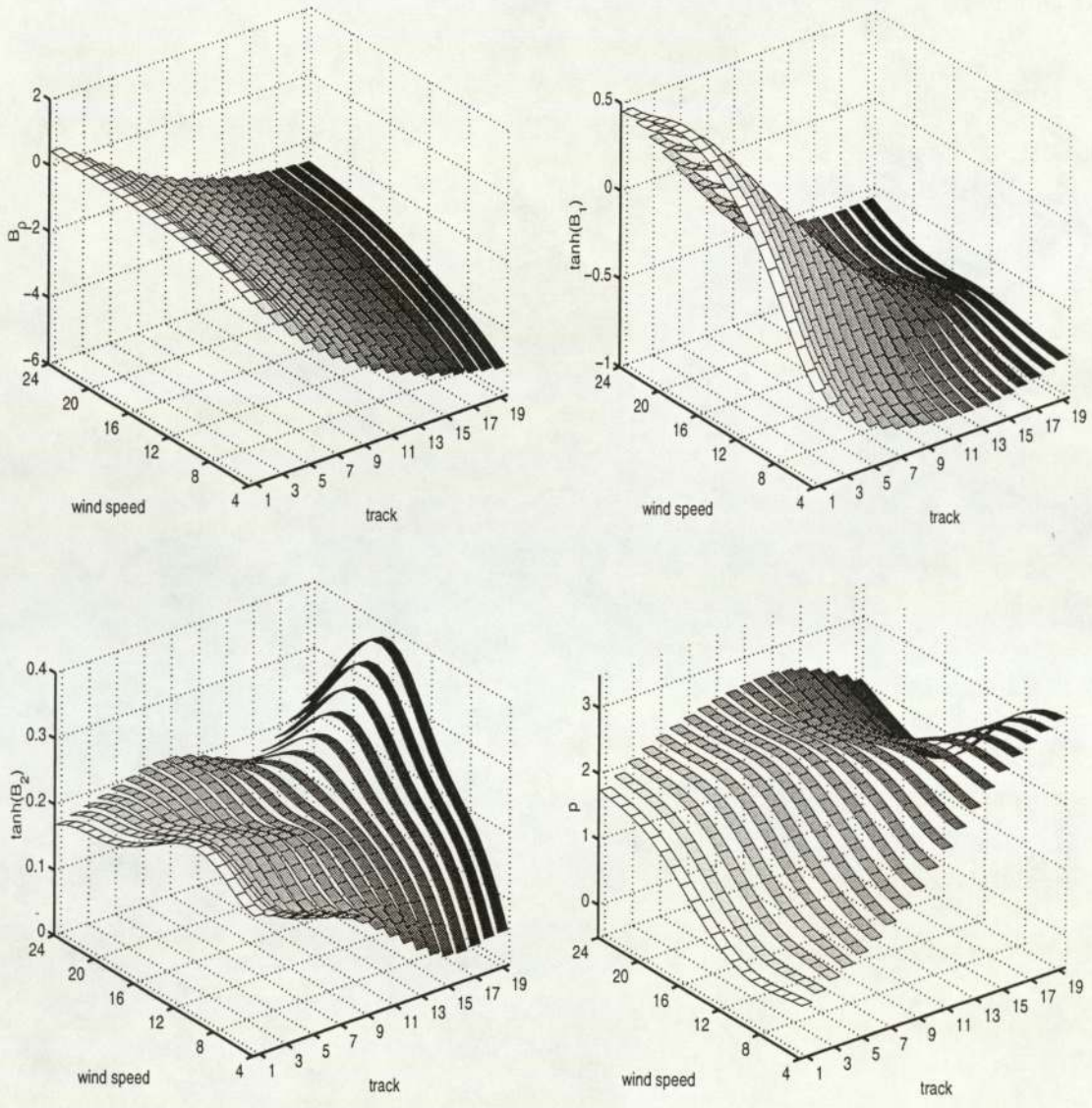


Figure 3.3: Outputs of the model NN2CMOD with 40 hidden units (one plot per output). Note that $B_0, \tanh(B_1)$ and $\tanh(B_2)$ cannot be considered as the coefficients of a Fourier Transform as the parameter p is varying.

Chapter 4

Graphical validation

This section lists the graphical representations which are useful for comparing the forward models. Some representations show σ^o measurements as a function of certain components (polar or Cartesian) of ECMWF winds. These plots can be misleading indicators of the quality of the forward model, because of the uncertainty in ECMWF winds.
However, a number of representations are plotted directly in σ^o space. As we know the targets lie close to a well-defined theoretical surface, these plots allow us to make sure the computed values of σ^o match this surface. This is the first step of a two-step validation proposed by (Stoffelen and Anderson, 1997b), the second step being a validation against winds, once the model is inverted.

4.1 σ^o as a function of wind speed and direction

An interesting representation appears in (Mejia *et al.*, 1998). It is the representation of the computed σ^o value as a function of wind direction where the incidence angle and the wind speed are fixed. Such plots are presented in Figure 4.1 for track 11 and for six different wind speeds. These plots show that all the models that have been developed and presented have similar features and they exhibit a large difference with CMOD4 and CMOD-IFR2. This representation shows that all the networks fit the clouds of points rather well. This is also true for most of the other neural networks which were trained with different numbers of hidden units (not shown here). For intermediate wind speeds, changing the complexity or the number of training iterations of the neural networks doesn't have any important

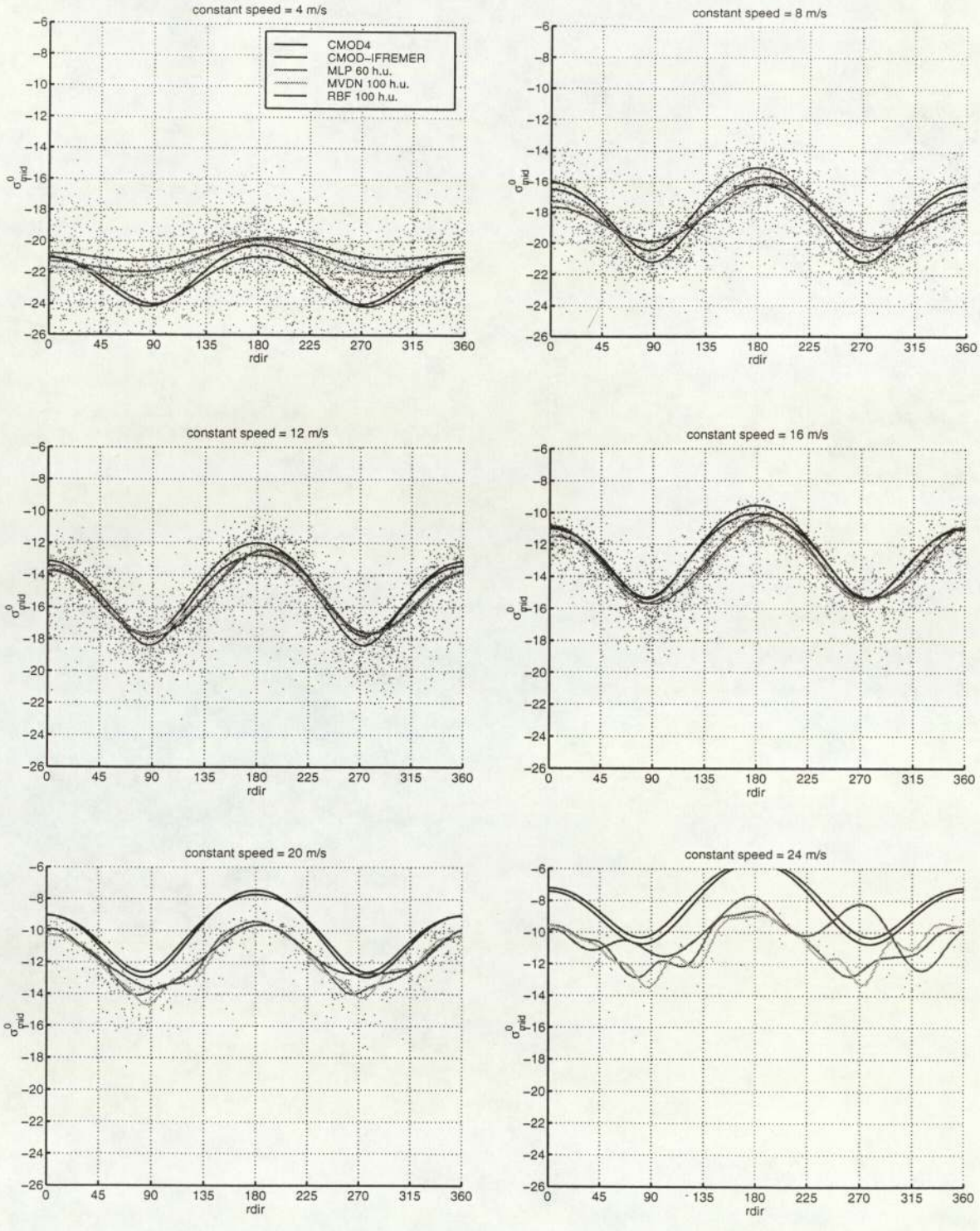


Figure 4.1: σ^o as a function of the wind direction. The thickness of the slice of selected points is ± 0.25 m/s. On these plots, all the models but CMOD4 and CMOD-IFR2 fit the cloud of points rather well. At high wind speeds where the training data is sparse, the neural networks outputs are very chaotic.

effect on the result. For high wind speeds where the data is sparse, the outputs of the neural networks are more and more chaotic as the number of hidden units increases. For CMOD4 and CMOD-IFR2, the extrapolation seems far better thanks to the restrictions imposed by their functional form. Moreover, high wind speed extrapolation for these models starts around 15-18 m/s.

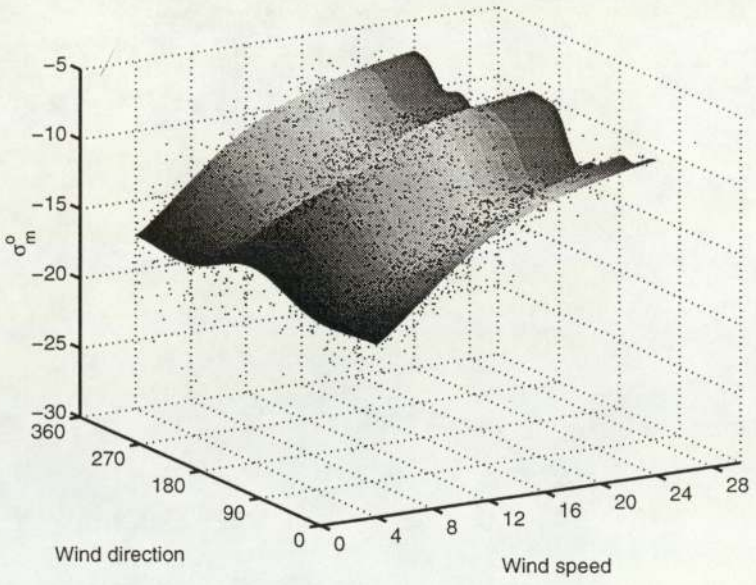


Figure 4.2: σ^o_m as a function of wind speed and wind direction, for track 11. The surface which is shown corresponds to the MLP with 60 hidden units. The colors represent wind speed.

An equivalent 3-D plot is the representation of $\boldsymbol{\sigma^o}$ as a function of both wind direction ϑ and wind speed $\|\boldsymbol{u}\|$ (Figure 4.2). The plots described above are just slices through this surface. Viewing this plot from above or from below shows that there are roughly as many points on each side of the surface for a model such as the MLP.

4.2 σ^o as a function of wind Cartesian components

An example of such a representation is shown in the introduction (Figure 1.3), where the surfaces are shown for different incidence angles. Actually, this plot could not be used for validation purposes.

However, it is quite helpful for the understanding of some of the main issues which will

arise when inverting the model, namely: the small dependency of σ^o on both wind speed and direction when the speed is high and the small range of values of σ^o for low incidence angles (below the satellite). The wind retrieval will be less accurate in these cases.

4.3 Vertical and cross-sections

Two graphical representations are proposed in (Stoffelen and Anderson, 1997a): a vertical section of the cone along the symmetry plane (of equation $\sigma_f^o = \sigma_a^o$) and a cross-section along the plane of equation $\sigma_f^o + \sigma_a^o = 2\sigma_{ref}^o$. Both of them are proposed in linear $\boldsymbol{\sigma^o}$ space. The speed is roughly constant in the cross-section.

The vertical sections of four different models are reproduced on Figure 4.3. They provide a powerful tool to compare the different models. The colours indicate the wind speed dependency. For the points, this speed is given by the ECMWF values. From these plots, it is clear that the RBF network has to be rejected. The MLP network is performing much better, with a smooth curve, although the curve does not fit the points perfectly. The expectation of the output of the MVDN has the same feature. Finally, CMOD4 has a better fit to the points for low wind speeds. However, above 15 m/s the fit is poor. Moreover CMOD4 will underestimate high wind speeds as the colours do not match. Unlike the other models, the two sheaths of CMOD4 have a common intersection with the vertical plane for wind directions of 45° and −45°. This is due to the definition of CMOD4 (it is symmetric).

The MVDN provides more information than the other models: it also gives the distribution of the $\boldsymbol{\sigma^o}$ triplets around the most probable value. As this distribution is assumed to be Gaussian, an isodensity surface is an ellipsoid in the $\boldsymbol{\sigma^o}$ space. The ellipses represent the intersection of the section plane with these ellipsoids for a few sample points (10 or so for each of the 4 intersection lines of the cone). The distance of the points of the ellipses from their respective centres is one (varying) standard deviation (see Appendix B.1).

The vertical section of the MVDN is presented in the log space. Indeed, the covariance matrices given by this model are valid in the log space. As the transformation is not linear, the isodensity surfaces are not easily representable in the measurement (*i.e.* linear) space as they become skewed ellipsoids.

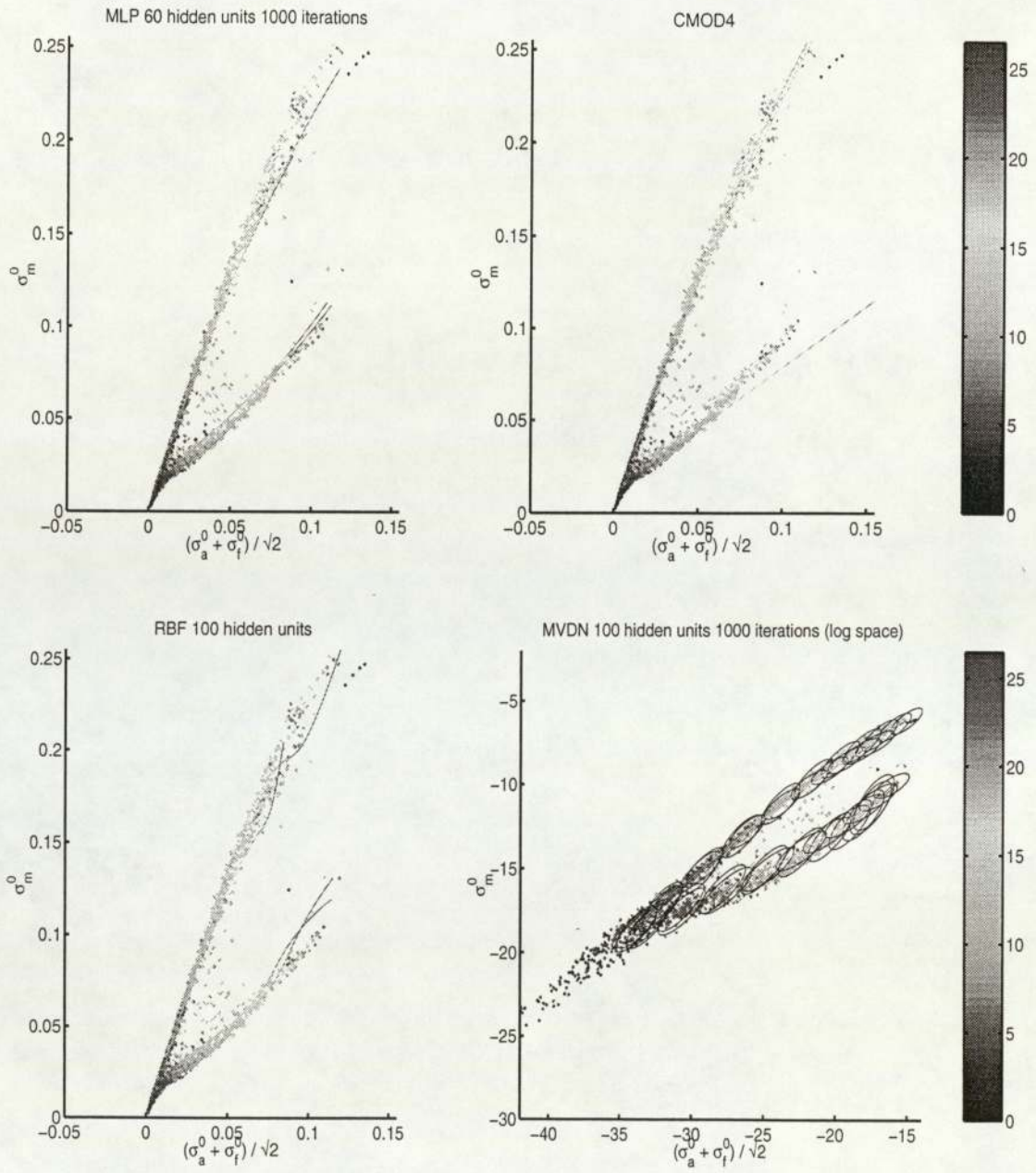


Figure 4.3: Vertical section of four models for track 11. The colour of the points refer to ECMWF speed. Each model is drawn for speeds up to 24 m/s. The MLP and the MVDN 'catch' the shape well but they do not exactly fit the points. CMOD4 doesn't fit the cross winds (lower limb). The RBF is poor although its bias is the smallest of all models (Table 2.3). MVDN: see text and Appendix B.1.

It can be noted that even for the MVDN, the selection of the points is performed in the measurement space (before transformation). The selected points satisfy the the equation $\sigma^o_f = \sigma^o_a + 0.018 * (\sigma^o_f + \sigma^o_a)$ for the cross-section. The equation of the section plane in the measurement space is $\sigma^o_{f,\text{lin}} = \sigma^o_{a,\text{lin}}$ which is equivalent to $\sigma^o_{f,\text{log}} = \sigma^o_{a,\text{log}}$, which is in turn the equation of the symmetry plane in log space.

Cross-sections of several models are shown in Figure 4.4. All the models but CMOD4 lie inside the 'theoretical' (target) cone. CMOD4 is symmetric: its functional form imposes that it takes the same values for ϑ and -ϑ. The equation of the cross section plane in the measurement space is $\sigma^o_{f,\text{lin}} + \sigma^o_{a,\text{lin}} = 2\sigma^o_{\text{ref}}$. Unlike the vertical section case, taking the log of left and right members of this equation no longer gives a linear relationship between $\sigma^o_{f,\text{log}}$ and $\sigma^o_{a,\text{log}}$. Therefore the cross section plane in linear space becomes a curved surface in log space and vice versa. The cross sections in log space are performed along the plane of equation: $\sigma^o_{f,\text{log}} + \sigma^o_{a,\text{log}} = 2\sigma^o_{\text{ref}}$ although it is not really perpendicular to the cone axis anymore.

4.4 Cone in target space

This section presents the most important tool in the process of developing and validating a forward model. The forward model is a function with three outputs and three inputs. One of the inputs (θ) takes discrete values so it can be fixed. The forward model is then a parameterised surface in $\boldsymbol{\sigma}^o$ 3D space with parameters ($\|\boldsymbol{u}\|$, ϑ). These input parameters can be shown as colors and/or lines on the surface of the cone. The $\boldsymbol{\sigma}^o$ measurements are plotted together with the cone. The associated values of ECMWF winds are not represented (unlike the vertical section). Many customisations can be brought to the program used to generate these plots (source code is in Appendix B.2). They actually give numerous interesting representations (see for instance figures on pages 56 and 78).

Figure 4.5 shows the $\boldsymbol{\sigma}^o$ measurements alone in the target space for a given track. The view is equivalent to a view 'from top' of the cone. One can easily distinguish the two sheaths as their distance is fairly constant in the log space. This implies that the output noise has a small amplitude (0.2 dB). The points at the bottom left in log space correspond to low wind speeds ($\lesssim$ 3 m/s), they do not appear in figures where the cones

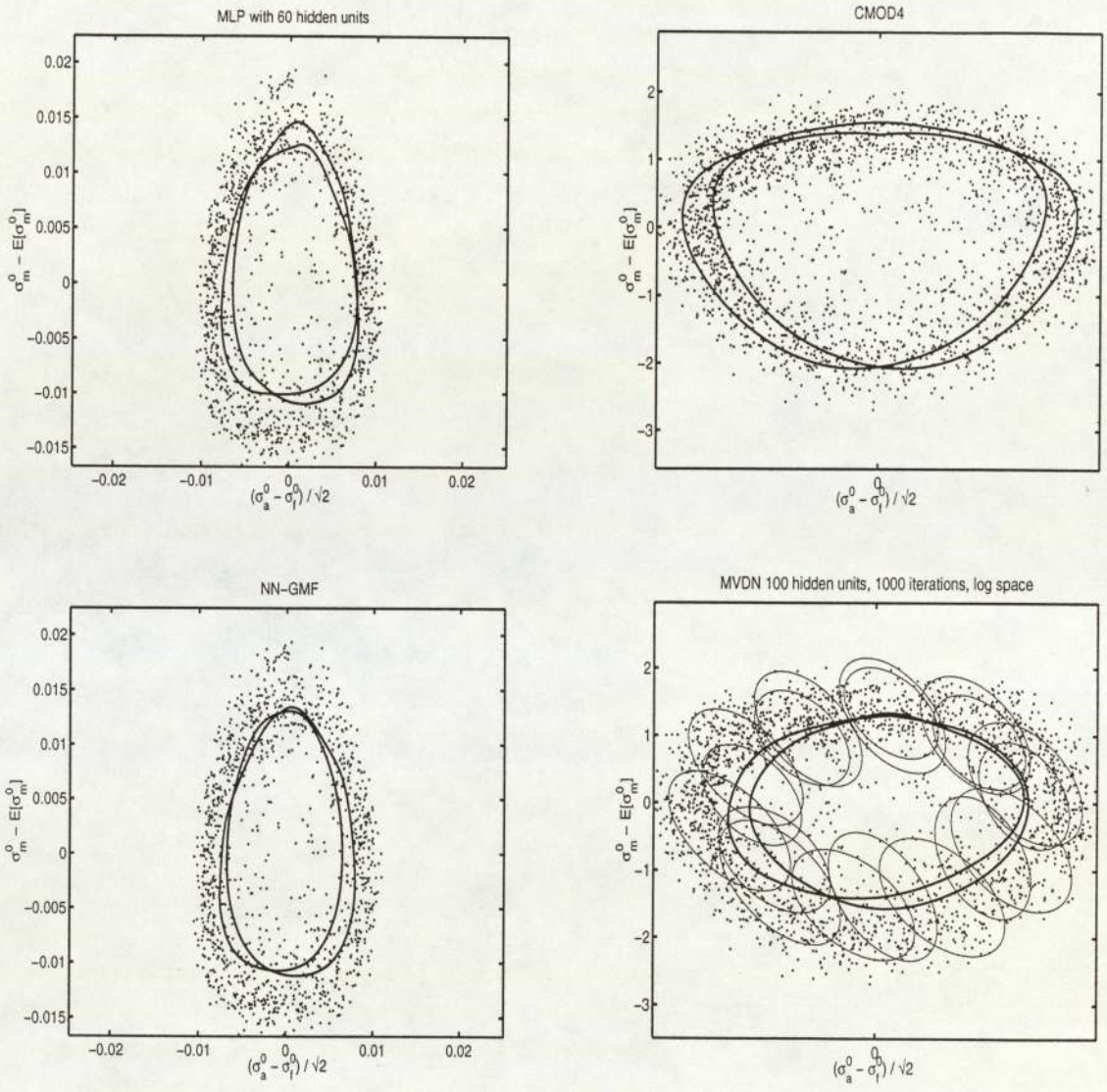
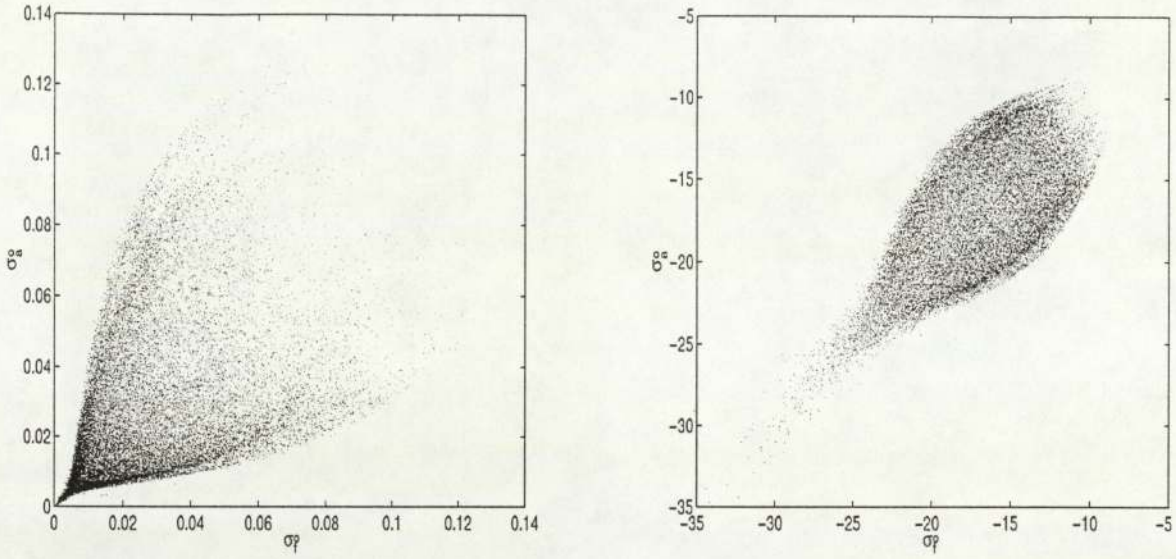


Figure 4.4: Cross section of four models for track 11. The average speed associated with the points is close to 8 m/s. The MLP and NN-GMF are very similar. The MVDN is plotted in log space, as is CMOD4 for comparison. CMOD4 best fits the data.



(a) Linear space

(b) Logarithmic space

Figure 4.5: Clouds of points of σ^o measurements for track 11 ($\theta \approx 34.9°$). The points are projected on the plane σ^o_m =0. About 30,000 points are represented here.

are represented. These clouds of points (one per track) are used in this study as test sets. Although the training data was subsampled from the corresponding files, not more than 4% of the points were selected. So these test datasets can be considered as independent from the training data.

In figures 4.6 and 4.7, the cones are displayed for speeds of 3-15 m/s (which corresponds to the speed range of the winds used to calibrate CMOD4 and NN-GMF) and 18-26 m/s (extrapolation for CMOD4 and NN-GMF). The black lines represent speeds of 6, 9 and 12 m/s on the cone. The colour coding is defined according to wind direction: red, yellow, green, purple represent 0, 90, 180 and 270° respectively. One quarter of the cone was removed ($\vartheta \in [135, 180]$ and $\vartheta \in [315, 360]$) to enable viewing the points and the inner sheath. These figures present "aerial views" of the cone which are the easiest to interpret. However, much more stringent selections between similar models can be obtained by comparing views "from top", where the the sheaths of points are most separated, as in Figure 4.5. For instance in Figure 5.1, one can still distinguish the two sheaths. Here is a comparison of the four presented models:

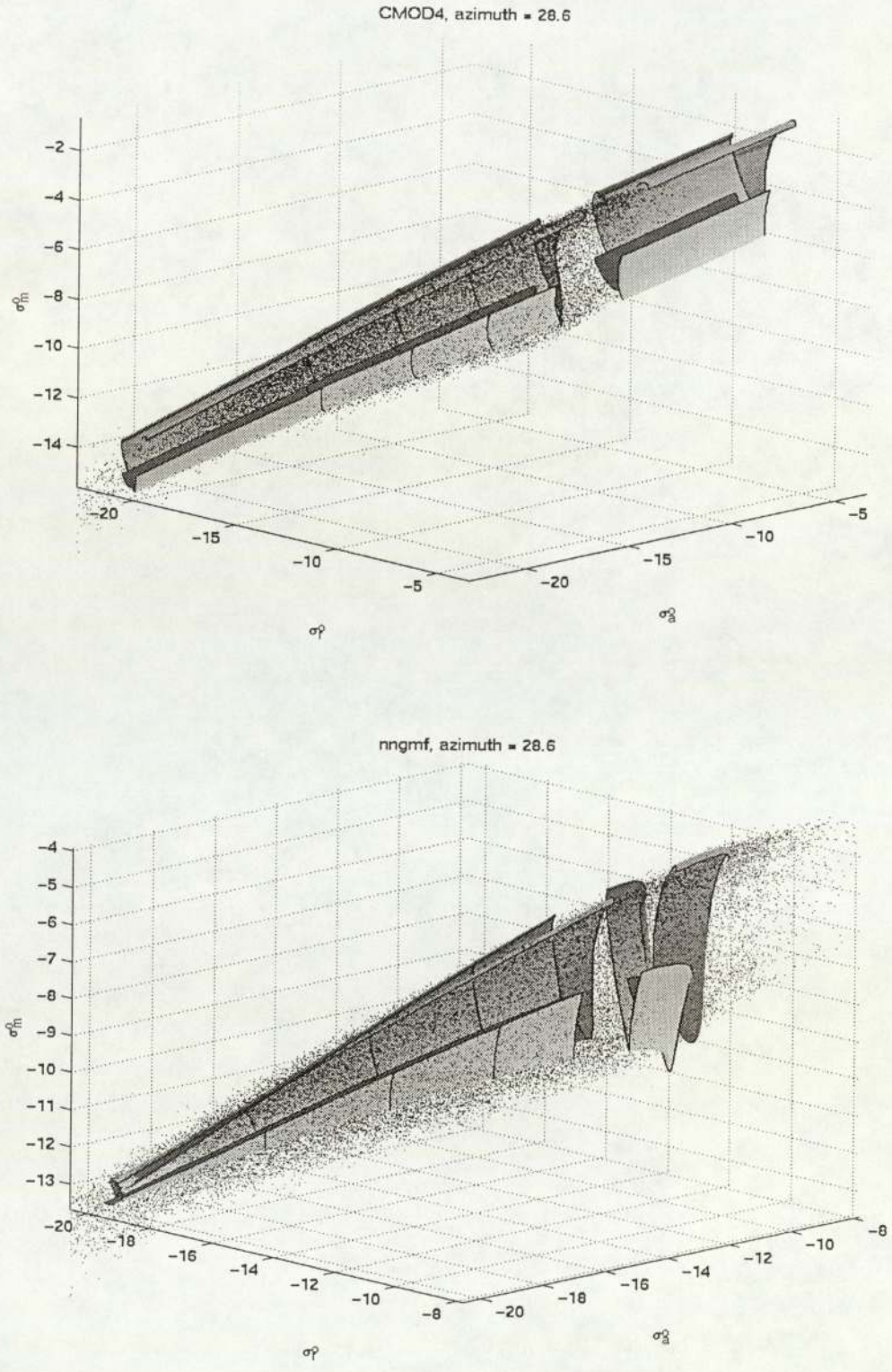


Figure 4.6: CMOD4 and NN-GMF, for track 9 ($\theta = 28.6$).

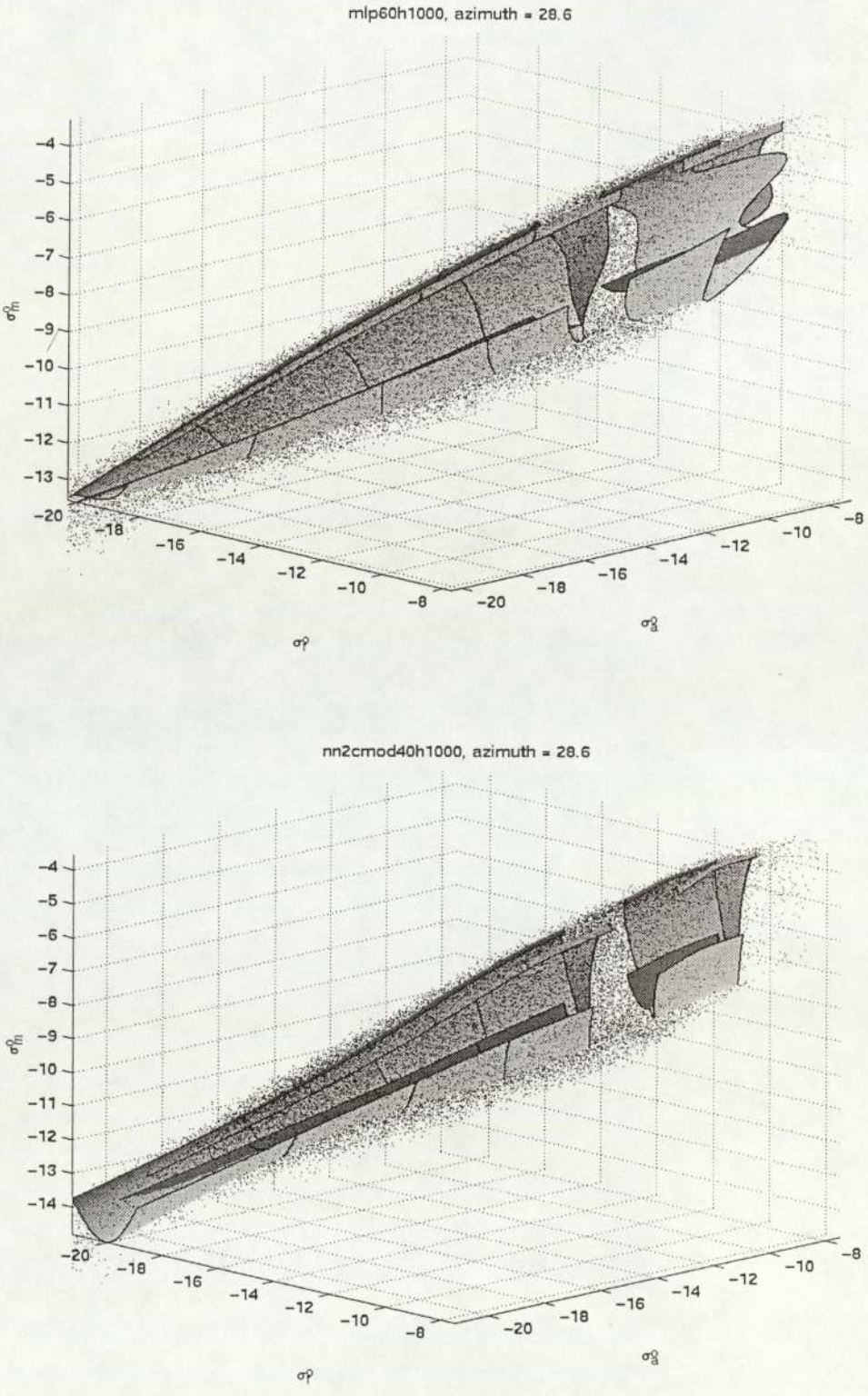


Figure 4.7: MLP with 60 hidden units and hybrid model, for track 9 ($\theta = 28.6$).

- The shape of CMOD4 is too close to a cylinder shape: at low and high wind speeds, the points lie far inside the surface. This had already been referred to in (Stoffelen and Anderson, 1995), it is just made clear here. On the contrary, at intermediate wind speeds, most of the points are outside of the surface. This may be due to a lack of free parameters or a restrictive functional form. And of course, the fitting is poor for high wind speeds because CMOD4 was not trained for these winds. CMOD-IFR2 exhibits very similar features (not shown). Also compare the position of the points in Figure 5.1 with the position of CMOD4 surface: CMOD4 retrieved speed is biased low at 10 m/s compared with ECMWF speed.

- For NN-GMF the fitting is poor. It is likely that the number of hidden units in the model is insufficient. The input noise was not taken into account.

- The MLP network with 60 hidden units catches the shape of the cone better, but its high number of hidden units makes the results very uncertain at high wind speeds. As the training data is sparse at high wind speeds, the model is overfitting.

- The last cone represents the hybrid model. Although the complexity is almost as high as the MLP above, the behaviour at high wind speeds is much better.

4.5 Conclusions

The graphical representations that are highlighted in this section are very helpful for the comprehension of many of the issues raised in this work. The view of the cone in σ^o space is the most important tool. It allows us to discard a number of models which do not fit the data. At this stage, CMOD4 and CMOD-IFR2 are among the best models. However, a model such as the MLP could be of interest. As it lies inside the theoretical cone, an autonomous disambiguation[1] will never be possible as the measured σ^o will always "project" onto the outer sheath of the model. But if the disambiguation method is efficient, then the determination of the wind vector may be of reasonable accuracy.

All the neural network models suffer from three major flaws:

[1]This term appeared in (Stoffelen and Anderson, 1997a): autonomous disambiguation is accomplished when the wind vector with highest MLE is the right solution.

1. They are not symmetric (see cross-sections of the cone). This is what one would naturally expect from the forward model because of the symmetry of the problem. The hybrid neural network (Section 3.2) was developed to correct this. In retrospect, it was found that maybe the forward model doesn't have to be symmetric: models where this is not a constraint seem to be in agreement about the asymmetry.

2. The neural networks give poor results at high wind speeds where the data is sparse or non-existent. NN-GMF has only 5 hidden units and it already catches the shape of the cone. This means there is room for reducing the complexity of the neural networks. It should remedy the problem.

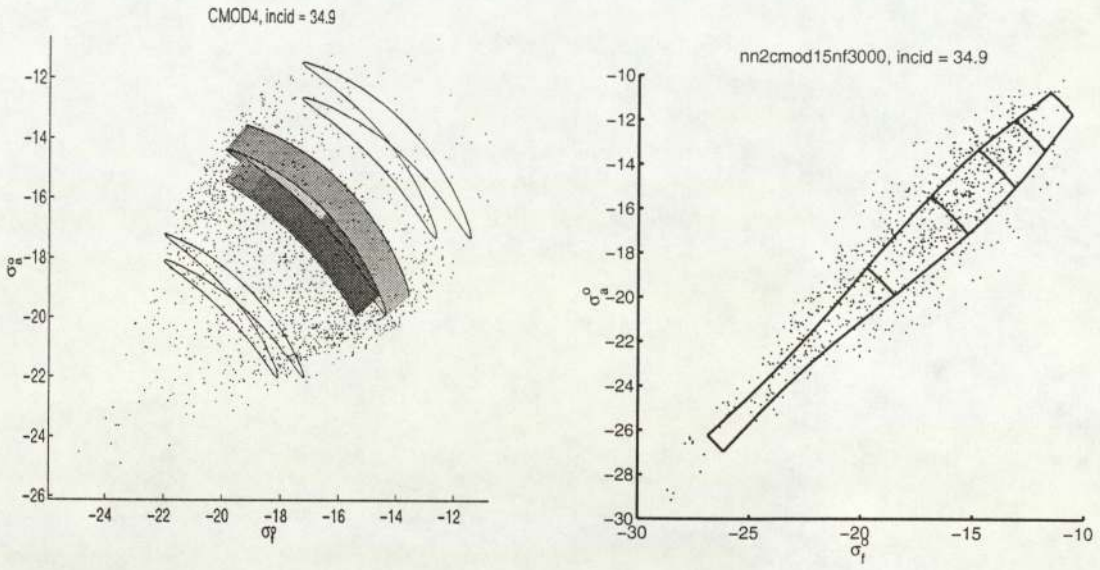
3. None of the neural networks gives outputs which fit the σ^o well. Reasons for this are given in Chapter 5.

Chapter 5

Training accounting for input noise

5.1 Evidence for input noise

Until now, the uncertainty in the inputs of the forward model has not been taken into account. As we know the scatterometer measurements lie on a surface parameterised by $(\|\boldsymbol{u}\|, \vartheta)$ for a fixed incidence angle, then a sub-selection of measurements on the basis of true (unknown) $(\|\boldsymbol{u}\|, \vartheta)$ should give a set of well gathered points. Figure 5.1 shows the measurements in $\boldsymbol{\sigma}^o$ space for track 11 and for an associated ECMWF wind speed range of 9-10 m/s. All the measurements of the file from which the points are sub-sampled are also shown in Figure 4.5 (compare the scales of the two figures: almost the whole cone could be drawn in Figure 5.1). As we can distinguish the inner from the outer sheath in Figure 4.5, this confirms the fact that the noise in $\boldsymbol{\sigma}^o$ space is small (around 0.2 dB). In Figure 5.1, we can see that the input noise in wind speed is responsible for a scatter of the points of roughly 5 dB. The scatter due to uncertainty in direction seems smaller. In any case, *the input noise cannot be neglected with respect to the output noise*, although this working assumption can often be made in the fields of neural networks and non-linear regression.



(a) Selection on wind speed

(b) Selection on wind direction

Figure 5.1: The points on the left plot show the measurements in σ^o space for track 11. They were selected for associated ECMWF wind speed range of 9 to 10 m/s. The surface is given by CMOD4 for the same range of speeds: it gives an idea of the size of the area where the points should lie in the absence of noise. For reference, the other lines represent 6 and 13 m/s on the cone according to CMOD4. Right plot: same thing but the selection criterion is $\vartheta \in [-10, +10°]$ and the model is nn2cmod (15 h.u., 3,000 iterations). Only the edges of the surface are represented.

5.2 Effects of input noise on the results

Here are the reasons why the conventional neural network methods for model selection fail to select the best models. The first method for model selection which was presented in this document consisted of comparing the σ^o measurements with computed σ^o values for the (noisy) ECMWF wind vectors which label the measurements. This method leads to bad conclusions. Indeed, we might expect a model to have RMS errors of around 0.2 dB. In Figure 2.5 (page 26), we can see that the errors lie in the range 0 to 5 dB, *i.e.* these errors are mostly due to input noise. All we can say about models which exhibit no bias in this test is that they are *good at modelling σ^o from noisy wind vectors*, but this is not what we want. For the same reason, using early stopping (this consists in stopping the training of the neural network if the test error starts increasing (Bishop, 1995)) or making any use of a validation error is not a valid method.

The positive bias at low σ^o values[1] ($\approx$ low wind speeds) comes from the cutoff in the test data at 4 m/s (see speed distribution of the test file on page 21). The lowest values of $\boldsymbol{\sigma^o}$ actually refer to true wind speeds around 1 m/s. But such low wind speed values do not appear in the ECMWF data we selected. So the corresponding low $\boldsymbol{\sigma^o}$ values (which may be correct by extrapolating the model) do not appear in the computed values of $\boldsymbol{\sigma^o}$, thus giving rise to positive bias. By contrast, in the intermediate speed range, the errors due to underestimated and overestimated ECMWF speeds cancel as the model was trained on this noisy input data, so the bias is nil. Nevertheless, CMOD4 and CMOD-IFR2 exhibit higher biases, which only means they are bad at predicting σ^o from noisy wind vectors. See also section 6.1 for deeper analysis.

In the same way, the plots on page 44 show that the networks learnt the noisy wind to $\boldsymbol{\sigma^o}$ transfer function. This transfer function is smoother than the true wind to $\boldsymbol{\sigma^o}$ function. In particular, as the $\boldsymbol{\sigma^o}$ dependency on wind speed is higher at low speed than at high speed (on page 44, the points are less and less scattered with increasing wind speed. Remember we are talking about noise along the 3$^{\text{rd}}$ axis), this dependency is underestimated so the neural network curves lie above the CMOD4 and CMOD-IFR2 curves at 4 m/s.

Let's consider the results for the probabilistic model for $P(\boldsymbol{\sigma^o} \mid u, v)$. Here again, the scatter in $\boldsymbol{\sigma^o}$ space which is modelled by Gaussian distributions comes from the effect of input noise on the distribution of the outputs. So the radii of the ellipses are about 5 dB along the cone generating line (wind speed uncertainty) and 1 dB perpendicularly along the surface of the cone (wind direction uncertainty). These values reflect the input noise, not the 0.2 dB antenna instrumental noise. Actually the noise in $\boldsymbol{\sigma^o}$ space is more likely to be spherical.

In $\boldsymbol{\sigma^o}$ space, the computed $\boldsymbol{\sigma^o}$ values always lie inside the theoretical surface on which the true measurements lie. This phenomenon is due to input uncertainty as shown in Figure 5.2. Indeed the surface is convex both along ϑ and $\|\boldsymbol{u}\|$ so the effects accumulate. Moreover, for a fixed angle, the dependency of $\boldsymbol{\sigma^o}$ on $\|\boldsymbol{u}\|$ is not linear so a model trained with noisy inputs will be biased, even if the fitting seems correct.

[1] Figure 2.5 shows the residuals for all tracks together, but computing the residuals for each track shows that $\boldsymbol{\sigma^o}$ is biased high at low wind speeds for each track.

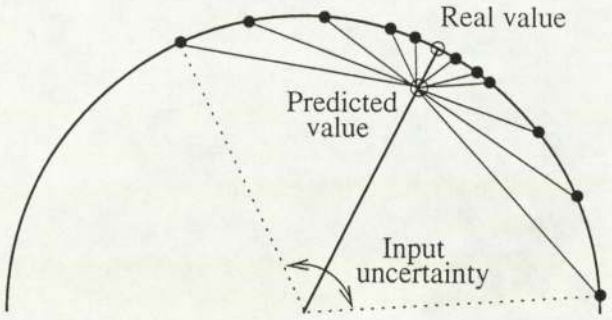


Figure 5.2: This sketch shows the effects of input noise on the prediction of σ^o by the forward model. Even a zero mean input noise will result in an output bias. This sketch can be considered as a cut along the cone as well as a cut across the cone as both speed and direction uncertainty will lead the surface defined by the model to lie inside the theoretical cone. Also note that the effect of wind direction uncertainty is doubled as the surface wraps twice.

5.3 Bayesian learning of neural networks

This section and the next one are mostly taken from a draft report by Dan Cornford, which in turn borrowed heavily from the ideas in a recent paper by Andy Wright.

Until now, we have considered a regression problem without input noise. If we consider the general regression problem then we have noisy inputs $\boldsymbol{z}$, true inputs $\boldsymbol{x}$, and noisy targets $\boldsymbol{t}$. The noisy targets are related to the true inputs by:

$$\boldsymbol{t} = y(\boldsymbol{x}; \boldsymbol{w}) + \boldsymbol{\varepsilon} \tag{5.1}$$

where $y(\boldsymbol{x}; \boldsymbol{w})$ represents the true transfer function (here written as a neural network depending on weight vector $\boldsymbol{w}$) and $\boldsymbol{\varepsilon}$ is the noise on the target, which is assumed Gaussian and independently and identically distributed, with zero mean and variance σ_t^2. This is what we want to learn, however we can only observe our noisy inputs, $\boldsymbol{z}$, and thus when training the network we will have to account for this.

Following Bishop (1995) we will first examine Bayesian learning in neural networks where the inputs are assumed noise free. We shall consider a very general solution based on sampling from the appropriate posterior distribution using Markov chain Monte Carlo (MCMC) methods (Neal, 1996), so that we can use the results in the following sections.

The problem we want to solve is the determination of $P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D)$ where $D = \{\boldsymbol{t_n}, \boldsymbol{x_n}\}, n = 1, ..., N$ is the training dataset, $\boldsymbol{x}^*$ is a new (unseen) input and $\boldsymbol{t}^*$ is the corresponding (unseen) output. With a neural network model the dependency of the target on the input is expressed through the weight vector $\boldsymbol{w}$ rather than the training set D. We can write this as:

$$P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D) = \int_{\boldsymbol{w}} P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, \boldsymbol{w}) P(\boldsymbol{w} \mid D)\, d\boldsymbol{w} \tag{5.2}$$

Now $P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, \boldsymbol{w})$ can be expressed as:

$$P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, \boldsymbol{w}) = \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp\left(-\frac{(y(\boldsymbol{x}^*; \boldsymbol{w}) - \boldsymbol{t}^*)^2}{2\sigma_t^2}\right) \tag{5.3}$$

assuming Gaussian errors on the target variable. We can rewrite $P(\boldsymbol{w} \mid D)$ using Bayes theorem:

$$\begin{aligned} P(\boldsymbol{w} \mid D) &= \frac{P(D \mid \boldsymbol{w}) P(\boldsymbol{w})}{P(D)} \\ &= \prod_n \left(\frac{P(\boldsymbol{t_n} \mid \boldsymbol{x_n}, \boldsymbol{w})}{P(\boldsymbol{t_n}, \boldsymbol{x_n})}\right) P(\boldsymbol{w}) \\ &\propto \left(\prod_n P(\boldsymbol{t_n} \mid \boldsymbol{x_n}, \boldsymbol{w})\right) P(\boldsymbol{w}) \end{aligned} \tag{5.4}$$

Now we can write:

$$P(D \mid \boldsymbol{w}) = \prod_n P(\boldsymbol{t_n} \mid \boldsymbol{x_n}, \boldsymbol{w}) = \frac{1}{(2\pi\sigma_t^2)^{N/2}} \exp\left(-\frac{\sum_{n=1}^{N} (y(\boldsymbol{x_n}; \boldsymbol{w}) - \boldsymbol{t_n})^2}{2\sigma_t^2}\right) \tag{5.5}$$

Now the only unspecified component in the model is the prior over weights $P(\boldsymbol{w})$. Here we will not choose the usual standard weight decay prior given by:

$$P(\boldsymbol{w}) = \frac{1}{(2\pi\sigma_w^2)^{W/2}} \exp\left(-\frac{\sum_{i=1}^{W} (w_i)^2}{2\sigma_w^2}\right) \tag{5.6}$$

where W gives the total number of weights in the network. Instead we will simply use

$$P(\boldsymbol{w}) = 1 \tag{5.7}$$

which means we don't use any regularization. This will be justified in the following.

Thus we now have the required distributions so that we can compute:

$$P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D) \propto \int_{\boldsymbol{w}} P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, \boldsymbol{w}) \prod_n P(\boldsymbol{t_n} \mid \boldsymbol{x_n}, \boldsymbol{w}) P(\boldsymbol{w})\, d\boldsymbol{w} \tag{5.8}$$

and thus we can sample from the posterior predictive distribution under very general conditions. In the above analysis we have considered a Gaussian error model for the target but if we use MCMC to sample from the posterior, then we can have very general noise and weight prior models. We can also calculate analytical solutions under simplifying assumptions using the Laplace approximation, but this is not treated here.

In practice we need to construct a Markov chain in $\{\boldsymbol{w}\}$. This is the training procedure of the neural network, it gives a set of vectors $\boldsymbol{w}$ sampled from $P(\boldsymbol{w} \mid D)$. For comparison, a frequentist approach only focuses on finding the most probable $\boldsymbol{w}$ from that posterior distribution. Using this set of vectors $\{\boldsymbol{w}\}$, we can compute the integral in Equation 5.8 using Monte Carlo integration. That is, we forward propagate each new input $\boldsymbol{x}^*$ through all the networks corresponding to the $\{\boldsymbol{w}\}$ to obtain samples of targets $\boldsymbol{t}^*$ drawn from $P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D)$. In order to obtain samples of $\boldsymbol{t}^*$ which are representative of $P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D)$, we need to ensure that the $\{\boldsymbol{w}\}$ are sampled from the stationary distribution, and independent. With this approach, we can also compute error bars around $\boldsymbol{t}^*$. This is not possible with a frequentist approach as only one value of $\boldsymbol{t}^*$ is computed. This value corresponds to $E[\, \boldsymbol{t}^* \mid \boldsymbol{x}^*, D\,]$.

5.4 Bayesian learning of neural networks with noisy inputs

This section considers the case where the inputs are noisy, that is:

$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{\zeta} \tag{5.9}$$

where we will assume that the input noise, ζ, is independently and identically distributed Gaussian with zero mean and variance given by Σ. Thus we have a generative input noise model:

$$P(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\Big\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{z})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{z}) \Big\} \tag{5.10}$$

We are now interested in the predictive distribution:

$$P(t^* \mid \boldsymbol{x}^*, D') \tag{5.11}$$

that is the predictive distribution of an unseen target with a new noise free input, whereas $D' = \{\boldsymbol{t}_n, \boldsymbol{z}_n\}$ is a noisy dataset. This is the model we would use to evaluate $P(\sigma_i^o | u_i, v_i, \theta_i)$ (as presented in Section 1.4). We consider the true inputs $\boldsymbol{x}_n$ to be latent (that is unobserved) variables, while we observe the noisy dataset. Thus during training of the network we must account for this input noise. We write:

$$P(t^* \mid \boldsymbol{x}^*, D') \propto \int_w P(t^* \mid \boldsymbol{x}^*, w) \int_{x_n} \prod_n P(\boldsymbol{x}_n, w \mid D')\, d\boldsymbol{x}_n dw \tag{5.12}$$

where this time we have to integrate out the dependency on the unobserved (latent) variables $\boldsymbol{x}_n$. If we again apply Bayes theorem then we can obtain:

$$\begin{aligned} P(\boldsymbol{x}_n, w \mid D') &= P(\boldsymbol{x}_n, w \mid \boldsymbol{t}_n, \boldsymbol{z}_n) \\ &= \frac{P(\boldsymbol{t}_n, \boldsymbol{z}_n \mid \boldsymbol{x}_n, w) P(\boldsymbol{x}_n, w)}{P(\boldsymbol{t}_n, \boldsymbol{z}_n)} \\ &= \frac{P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, w) P(\boldsymbol{z}_n \mid \boldsymbol{x}_n) P(\boldsymbol{x}_n) P(w)}{P(\boldsymbol{t}_n, \boldsymbol{z}_n)} \\ &\propto P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, w) P(\boldsymbol{z}_n \mid \boldsymbol{x}_n) P(\boldsymbol{x}_n) P(w) \end{aligned} \tag{5.13}$$

where we use the fact that conditionally on $\boldsymbol{x}_n$:

$$P(\boldsymbol{t}_n, \boldsymbol{z}_n \mid \boldsymbol{x}_n, w) = P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, w) P(\boldsymbol{z}_n \mid \boldsymbol{x}_n) \tag{5.14}$$

and the dependency on the weights in $P(\boldsymbol{z}_n \mid \boldsymbol{x}_n, \boldsymbol{w})$ has been removed. It can also be seen that we now require $P(\boldsymbol{x}_n)$, an unconditional (true) input density model. The distribution of the samples in the noisy dataset is known, so we can specify $P(\boldsymbol{z}_n)$. Then $P(\boldsymbol{x}_n)$ can be found by making use of Equation 5.9. We can now write:

$$P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, D') \propto \int_{\boldsymbol{w}} P(\boldsymbol{t}^* \mid \boldsymbol{x}^*, \boldsymbol{w}) \int_{\boldsymbol{x}_n} \prod_n P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{w}) P(\boldsymbol{z}_n \mid \boldsymbol{x}_n) P(\boldsymbol{x}_n) P(\boldsymbol{w}) \, d\boldsymbol{x}_n d\boldsymbol{w} \tag{5.15}$$

Thus it can be seen that during sampling the Markov chain we will need to run over $\{\boldsymbol{x}_n, \boldsymbol{w}\}$. Fortunately, we don't need to store the samples of $\{\boldsymbol{x}_n\}$: instead we can store for each sample of $\boldsymbol{w}$ the corresponding value of

$$\int_{\boldsymbol{x}_n} \prod_n P(\boldsymbol{t}_n \mid \boldsymbol{x}_n, \boldsymbol{w}) P(\boldsymbol{z}_n \mid \boldsymbol{x}_n) P(\boldsymbol{x}_n) P(\boldsymbol{w}) \, d\boldsymbol{x}_n \tag{5.16}$$

However, the size of the dataset should be reduced as much as possible in order to keep the Markov chain at a reasonable size.

5.5 Practical implementation

5.5.1 Application to the wind retrieval problem

As the nn2cmod model had the best features so far, this model was chosen to be trained with Bayesian learning. The first thing to do is to relate the notation above to the notation of the wind retrieval problem, and give a context to the terms in equation 5.16.

The targets $\boldsymbol{t}_n$ are the triplets of σ^o measurements and n is running over the number of patterns in the dataset. The inputs $\boldsymbol{x}_n$ are the wind vectors components (u, v) and the incidence angle θ. These three inputs play very different roles: θ is not subject to any uncertainty, it is just a fixed parameter. Therefore it is not sampled in the Markov Chain. However, it is necessary to keep track of this parameter as the noisy inputs and the corresponding inputs in the Markov chain must be labelled with a common value for θ. To keep a reasonably sized Markov chain, the size of the training set is decreased to 1,000 patterns. The number of hidden units of the nn2cmod model is set to 10. This makes

$2 * 1,000 + 74 = 2,074$ parameters in the Markov chain, which is still a very large number.

5.5.2 Description of the energies in the Markov chain

As the distributions in Equation 5.16 are Gaussian, it is more convenient and more efficient to minimise the energy associated with the probabilities rather than to maximise the probabilities themselves. Four energies are defined as follows:

$$
\begin{aligned}
E_1 &= -\ln\Big(\prod_n P(\boldsymbol{t_n} \mid \boldsymbol{x_n}, \boldsymbol{w})\Big) \\
E_2 &= -\ln\Big(\prod_n P(\boldsymbol{z_n} \mid \boldsymbol{x_n})\Big) \\
E_3 &= -\ln\Big(\prod_n P(\boldsymbol{x_n})\Big) \\
E_4 &= -\ln(P(\boldsymbol{w}))
\end{aligned}
\tag{5.17}
$$

The total energy of the system is $E = E_1 + E_2 + E_3 + E_4$. We now switch to the wind notations. $(u_{\mathrm{MC}}, v_{\mathrm{MC}})$ are the samples in the Markov chain. (u, v) are the corresponding values in the training set.

$E_1 = -\ln\big(\prod_n P(\boldsymbol{\sigma^o} \mid u_{\mathrm{MC}}, v_{\mathrm{MC}}, \theta, \boldsymbol{w})\big)$ is the error of the model, calculated for the given set of satellite measurements and for (u_{MC},v_{MC}) which will hopefully tend to be a noise free set of wind components during training.

$E_2 = -\ln\big(\prod_n P(u_{\mathrm{MC}}, v_{\mathrm{MC}} \mid u, v)\big)$ is the error due to the wind vectors in the Markov chain going away from their associated ECMWF winds.

$E_3 = -ln\big(P(u_{\mathrm{MC}}, v_{\mathrm{MC}})\big)$ only plays a small role here. It associates some error to the winds in the Markov chain which are not likely to be derived from the dataset.

$E_4 = -ln(P(\boldsymbol{w})) = 0$ We don't want E_4 to play any role. Three factors are encouraging us not to use weight decay regularization here. First, we know our forward model is already too smooth. Secondly, the outputs of the MLP in the nn2cmod model which will be trained are not normalised. Thus a centred Gaussian prior over all the weights is not a very good assumption[2]. Third, it is faster to compute and simpler to implement.

[2] A more complicated weight prior could be used, but at this stage, it is not worth investigating it.

Mathematical expressions

The expression of the error of the model E_1 has already been given in Section 3.2. It is recalled here:

$$E_1 = \frac{1}{2\sigma_{\sigma^o}^2} \sum_n (\sigma_n^o - \sigma_{n,t}^o)^2 \tag{5.18}$$

Following (Stoffelen and Anderson, 1997b), the error E_2 of the wind vectors was computed in (u, v) space, with Gaussian error distributions. If we denote by $\sigma_{u,v}$ the common standard deviation of the errors of ECMWF winds in (u, v) components, we can write:

$$E_2 = \frac{1}{2\sigma_{u,v}^2} \sum_n \left[(u - u_{\mathrm{MC}})^2 + (v - v_{\mathrm{MC}})^2 \right] \tag{5.19}$$

$\sigma_{u,v}$ was set to 2.0 m/s which is higher than the recommended value of 1.5 m/s found in (Stoffelen and Anderson, 1997b). From Figure 5.1, it seems 1.5 is an underestimate. Anyway, this parameter is difficult to evaluate.

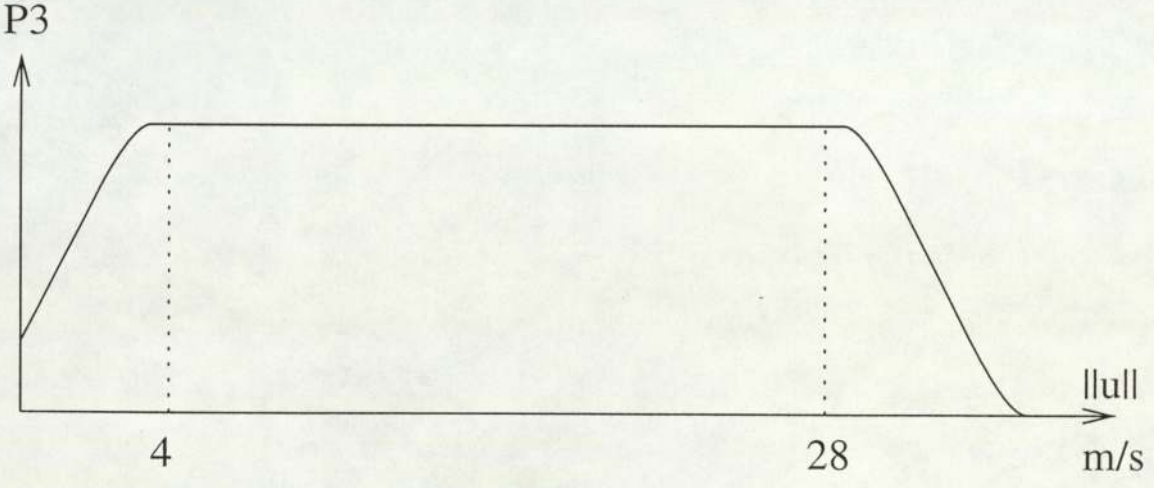


Figure 5.3: Probability distribution $P_3 = P(\|\boldsymbol{u}\|)$. It is uniform between 4 and 28 m/s which are the cutoff values in the training set. Beyond these values, the probability decreases exponentially.

E_3 was defined by parts: as the selection of the training set was performed in order to have a roughly uniform wind direction distribution and a set of speeds between 4 and 28 m/s, E_3 was taken to be a function of speed only. The form of the associated probability $P(\boldsymbol{x_n}) = P(\|\boldsymbol{u}\|)$ is shown in Figure 5.3. As stated above, this error was found to

play a very small role as the errors E_1 and E_2 constrain the speed to be in the range of speeds that are present in the dataset anyway.

Choice for a starting point

There are 2074 parameters in the Markov chain. This means the algorithm will perform a random walk in a space of dimension 2074, looking for a minimum of the energy function. Choosing a good starting point is fundamental if we do not want to face months of computational time. For the vector of weights, we have a very good candidate for a starting point, that is the set of weights of the network which was trained without accounting for the noise. For the wind vectors, a natural solution is to start from the (u, v) vectors in the dataset. This option was adopted at first but it was rapidly replaced by the (u, v) values given by an inverse model of wind retrieval. This model was being developed by David Evans at the NCRG. Figure 5.4 gives a good idea of how far these values are from NWP winds. It is to be noted that the inverse model was trained with winds up to 24 m/s only. Plots such as Figure 5.4 showed that NWP winds even above 23 m/s were systematically pulled toward lower speeds (this was confirmed by comparing the different initial errors of the Markov chain), hence the choice for a threshold of 23 m/s. No particular feature appears in Figure 5.4 (no speed or direction error dependency, no clusters in the outputs), which is encouraging for the quality of the starting point.

Training

Once the noise levels in the input and output spaces have been chosen, two parameters have to be tuned in the Metropolis algorithm. These are the two step sizes in the Markov chain. They have to be different for the proposals in (u, v) space on the one hand and in $\boldsymbol{w}$ space on the other hand. By comparing their ranges of values, the ratio of the two step sizes was estimated to be around 10. But this ratio should also reflect how confident we are about the weights and the wind vectors: the starting point in the weight space is known to be good, so we want the (u, v) components to move faster than the weights at first.

Actually, after a few trials, it was found that these step sizes were very difficult to

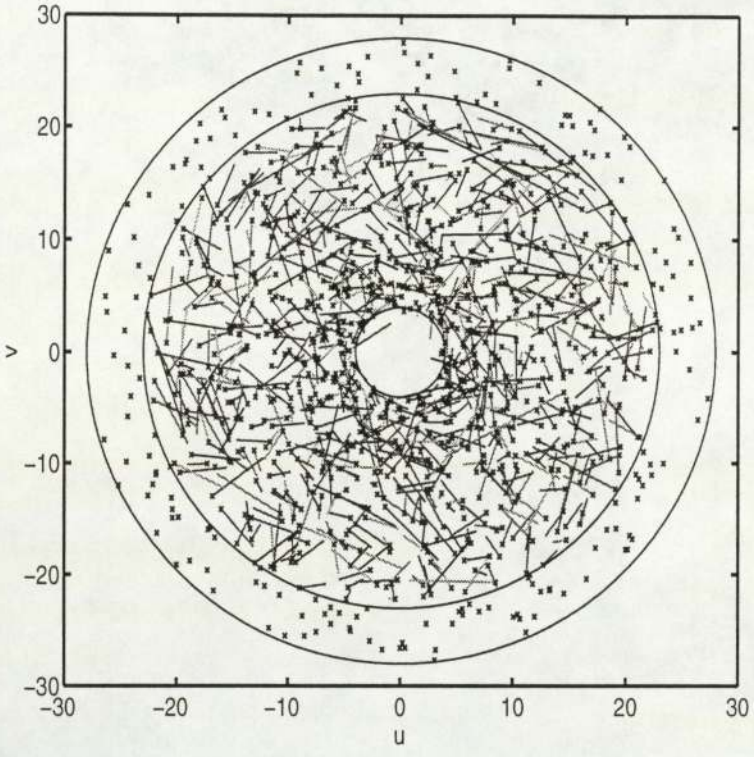


Figure 5.4: Initialisation of (u, v) components for Bayesian learning. The crosses represent NWP winds, lines link them to the initial state for the Markov chain. Circles represent speeds of 4, 23 and 28 m/s. All incidence angles are present. Colours are for ease of reading only.

choose. Allowing the weights to move always made the error E_1 go uphill ! Indeed, the moves were accepted as E_2 was going downhill much faster. Graphically, the fit to the cone was getting worse and worse very quickly. So the step size in $\boldsymbol{w}$ was set to almost zero most of the time during these trials.

A stationary distribution of the parameters was never reached: after several tens of hours of running during which the the step sizes were changed, the algorithm was still in the burn-in stage. That is to say it was just performing a very slow descent toward the minimum. Moreover, it is to be noted that if large moves had been allowed for (u_{MC},v_{MC}), then there would have been a risk of falling into one of the numerous local minima associated with each ambiguous solution. A procedure was implemented in order to ensure the winds were not approaching the opposite direction from ECMWF values, but no such winds were

detected.

In the view of the difficulties above and fearing that only 1,000 points would not be enough to get proper results, this approach was abandoned. Instead, a gradient descent was performed. This technique does not allow for modelling probabilistic distributions in σ^o space as Bayesian learning does, but this is not really feasible in such a high dimension. Even in the event of finding a stationary distribution using Bayesian learning, making the samples independent would require a very big computational effort.

5.6 Gradient based optimisation

In the previous section, a Markov chain was found to be incapable of approaching a minimum of the error function in a very high dimensional space.

The derivatives of the error functions on which we were sampling were computed in order to perform a non-linear optimisation (using the SCG algorithm). The notation for E_1, E_2 and E_3 is the same as above.

As the inputs to the nn2cmod model were actually $(\|\boldsymbol{u}\|, \vartheta)$, these parameters were chosen as inputs rather than (u, v). So the parameters to optimise were $(\|\boldsymbol{u}\|, \vartheta, \boldsymbol{w})$.

The following derivatives were calculated:

$$\frac{\partial E_i}{\partial \vartheta}, \quad \frac{\partial E_i}{\partial \|\boldsymbol{u}\|}, \quad \frac{\partial E_i}{\partial \boldsymbol{w}}, \qquad i = 1...3 \tag{5.20}$$

Their expressions are given in Appendix C.

Using these derivatives, the nn2cmod model with 10 hidden units was trained again using the SCG algorithm. After a hundred iterations only, the fitting of the cone had drastically improved for most incidence angles; the cone was obviously inflating. Unfortunately, it was still far from being perfect and the progression was slow in the following iterations. In fact, the weights had quickly adjusted to fit the new data (from David Evans' inverse model) but the $(\|\boldsymbol{u}\|, \vartheta)$ variables were moving very slowly. Plots such as Figure 5.4 showed that the optimised directions almost didn't change compared with initial values. This problem was rectified by normalising the variables ϑ and $\|\boldsymbol{u}\|$ ($\boldsymbol{w}$ was already in the range [-1, 1]). The derivatives of the errors were updated as necessary. The

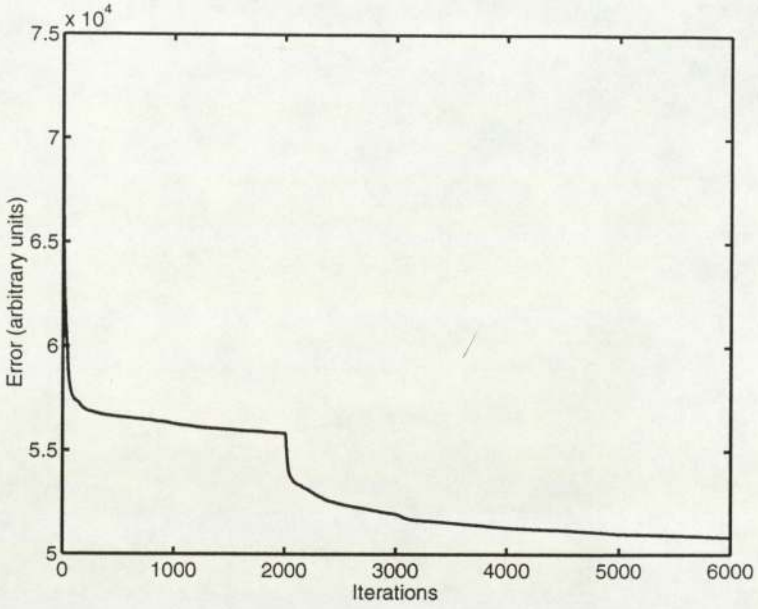


Figure 5.5: Learning curve for nn2cmod with 20 hidden units, trained using the SCG algorithm. At 2,000 iterations, the parameters to optimise were normalised (see text).

effect of this normalisation is shown in Figure 5.5.

After a few thousand training iterations, the fitting in σ^o space is very good (Figure 5.6), except for low wind speeds. This is certainly due to data selection (Section 6.1). With 10 hidden units only for the MLP in nn2cmod, straight lines appear in some areas of the generating lines of the cone in σ^o space (not shown). A complexity of 20 hidden units seems too high as the behaviour of the surface for high wind speeds changes too quickly as soon as the data is sparser. 15 hidden units seems to be a reasonable compromise.

The variance of the errors in ECMWF winds is another important parameter. It was set to 2 m/s. Smaller values could be tried as long as the fitting in σ^o space is good. They would diminish some risk of fitting well while having biased retrieved winds. In the case of a very large error level specification, the fitting could be excellent but the relationship between $(u_{\mathrm{MC}},v_{\mathrm{MC}})$ and (u,v) would be lost, thus leading to poor retrieved winds. These effects are difficult to detect and no other values were investigated for the input noise level specification.

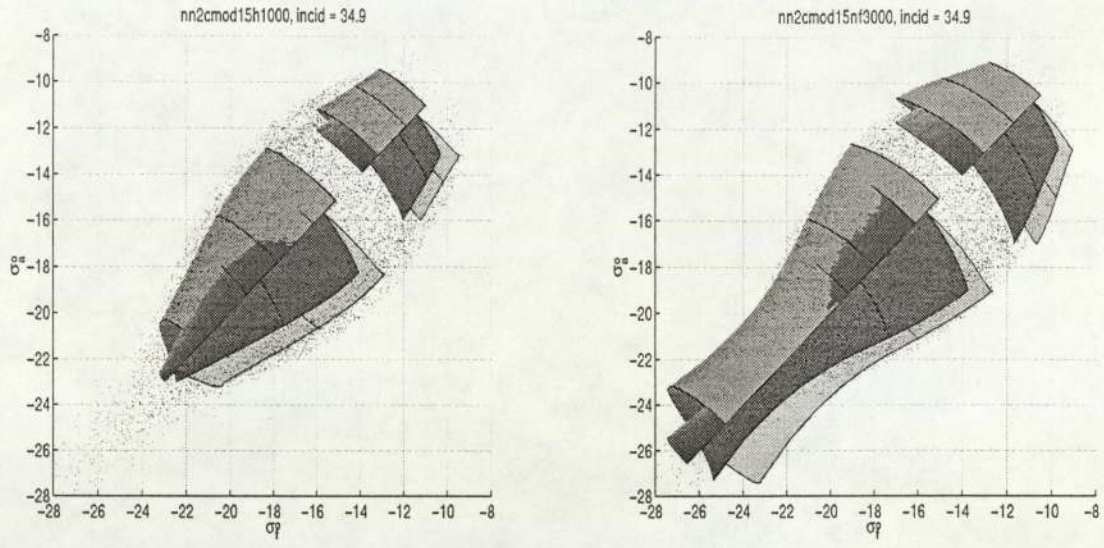


Figure 5.6: nn2cmod trained accounting (right plot) / not accounting (left plot) for input noise. View "from top" of the cone, track 11.

5.7 Conclusions

In this chapter, it was proved that the uncertainty in the ECMWF wind vectors cannot be neglected. For the forward model, this input noise makes the conventional use of test and validation errors inappropriate to validate the various neural networks. Models with similar RMS errors can be very different.

During the training procedure, the input noise has an obvious effect; in target space, the surface defined by the model always lies inside the target cone. Secondly, the input noise will lead to biased retrieved speeds as the dependency of σ^o on speed is not linear.

A Bayesian framework is adopted to account for input noise during the training procedure of the neural network. Bayesian learning can determine input and output noise levels. It can also give error bars for the outputs. Unfortunately, this technique is too computationally intensive, it has to be abandoned.

Following this observation, a gradient descent is used to find the Maximum A Posteriori of the posterior distribution of the parameters of the Markov chain. The computational effort is now reasonable. This technique leads to a model which fits the data well in target space, but it may still be suboptimal as the input and output noise levels are determined

empirically. Anyway the obtained model is good enough so we can now consider its inversion. This is addressed in the next chapter.

Using the Maximum A Posteriori of the posterior distribution of the parameters of the Markov chain as a starting point for this Markov chain, it may be possible to reduce the computational effort of Bayesian learning. This is not investigated here.

Chapter 6

Validation against winds and improvements

In Chapter 5 the fitting of the surface of nn2cmod in σ^o space was sufficiently improved enabling a validation against winds for this model. A good fitting does not necessarily mean a good quality of wind retrieval.

6.1 Another effect of input uncertainty

It has already been shown that using noisy inputs in the training procedure affects the fitting in σ^o space; it can also lead to biased retrieved speeds. There is another effect of input noise; it plays a role during the data selection. It has a strong impact on the model which is trained on that data, even if it is trained accounting for input noise.

By comparing plots of the 3D cone in σ^o space for some nn2cmod models (with 10, 15, 20 hidden units, trained accounting for noise) on the one hand with the cone for CMOD4 on the other hand, one can infer that the predictions of wind speeds at a low wind speed range are much higher for nn2cmod models than for CMOD4. This is revealed by the surface which extends much further toward low wind speeds for nn2cmod. Although it is recognised in the literature (Janssen *et al.*, 1998) that CMOD4 predictions are too low for low wind speeds, the difference is surprisingly large.

So what is happening? The reason is made fairly clear in Figure 6.1. The dataset we

are using was filtered on the basis of ECMWF noisy wind speeds. All the σ^o measurements labelled with speeds under 4 m/s were discarded from the dataset. This was originally intended to improve wind retrieval for winds above 4 m/s as the backscatter process simply doesn't work (measurements are very noisy) at speeds below this threshold. Actually this selection is very arbitrary. In the resulting dataset, σ^o measurements corresponding to true low wind speeds are still present, but all of them are labelled by ECMWF with overestimated speeds above 4 m/s. Thus the errors in the dataset are biased so the assumption of a centred Gaussian noise distribution no longer holds. In Figure 6.1, a Gaussian[1] noise with zero mean and standard deviation 1.5 was added to a set of uniformly distributed 'true' speed values between 0 and 14 m/s. This noise results in a bias in the selected dataset for winds up to 7 m/s. Moreover, the same effect will arise from a cutoff at high wind speeds, and any selection criterion applied to the data according to ECMWF wind speeds will introduce biased errors. Such methods appear in several articles in the literature.

Any model which is trained on this data will learn the biased regression. This is revealed by the 3D cone as explained above. It is also revealed by Figure 5.4 for the inverse model by David Evans which was trained on similar data. Figure 5.4 shows that all predicted values lie above 4 m/s although it would be expected that some of them should lie below. This also explains (to some extent) why David Evans had to use several superimposing Gaussian kernels to model the error distributions as they are skewed at wind speeds just above 4 m/s.

The following conclusions can be drawn:

- One must not subsample the data on the basis of ECMWF speed. Using a threshold as above has the worst repercussion on error bias. Instead, the natural distribution of wind speeds could be kept. This constraint rehabilitates the idea of using a cost function (Cornford and Nabney, 1998a) if we want to give greater importance to the errors that occur for (rare) high wind speeds. A second alternative is to select the samples on the basis of a classification in σ^o space by geometrical means. This is a safe method as the density of the inputs doesn't affect the output noise distribution.

[1]This noise is not supposed to be Gaussian and this is particularly wrong with decreasing wind speed, but the approximation is good enough for this artificial problem.

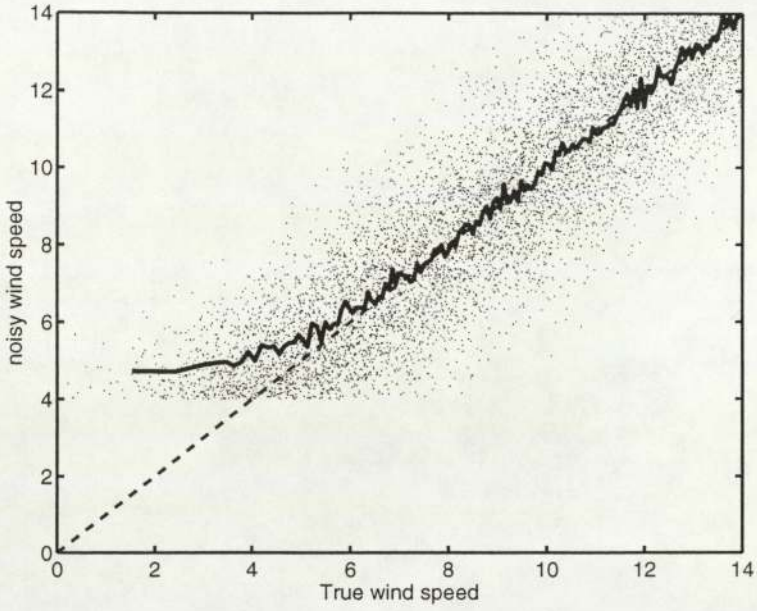


Figure 6.1: The effect of input noise during data selection. Selecting data with a threshold of 4 m/s on the basis of noisy wind vectors introduces a positive bias (up to 5 m/s) on winds between 0 and 8 m/s. This can be read on this plot by comparing the height of the dashed line and the solid line (vertical moving average of the selected points). The assumption of a centred Gaussian noise distribution in (u, v) components no longer holds.

The $\boldsymbol{\sigma^o}$ points can be classified using planes of equation $\sigma_a^o + \sigma_f^o = 2\sigma_{\text{ref}}^o$ as speed is roughly constant in these planes. The values for σ_{ref}^o must be defined at regular intervals of speed for each track. There are two advantages for this geometrical method. First, the error bias discussed above will be removed. Secondly, it is possible not to select points in the area in $\boldsymbol{\sigma^o}$ space where the spreading of the points is large (by graphical means). Thus the model itself will determine the corresponding speed threshold. It can be noted that many of the points selected using a (noisy) speed threshold of 4 m/s actually lie in areas of $\boldsymbol{\sigma^o}$ space where the spreading is large (see figure 4.5(b)). It is not worth training the model with these points.

- As far as the work presented in this document is concerned, our position about regularization may need to be revised. Indeed, during training, nn2cmod fits better and better to only a few points which correspond to low wind speeds and which

are labelled with systematically overestimated ECMWF wind speeds. Weight decay may avoid this phenomenon in the absence of proper data. Indeed, the base of the cone surface becomes very flat as a wide range of target points are labelled with an ECMWF speed just above 4 m/s. Thus the σ^o-to-speed mapping is very steep in that region (see the 'stack of funnels' for nn2cmod, page 12). In a neural network, some of the weights need to move toward large values to obtain a very steep tanh function. So weight decay may prevent the mapping from being very steep. Of course, the use of well selected data is the best solution. After a few thousand more iterations of the SCG algorithm, the nn2cmod model with 15 hidden units over-fits the data at high wind speeds anyway (this is why the reference nn2cmod model in this document is not trained for more than 3,000 iterations). So weight decay is useful, even with well selected data.

The other conclusion about this work is that $P(u_{\mathrm{MC}}, v_{\mathrm{MC}})$ (it is shown in Figure 5.3) must be badly defined as it discourages 'optimised' winds to go down to wind speeds below 4 m/s. With a well selected dataset, this prior should be used to prevent optimised winds to go down to negative speeds. Actually, this prior should represent the natural atmosphere speed distribution. This will also correct bias in ECMWF wind speeds. ECMWF wind speeds have to be biased[2] as the natural atmosphere speed distribution is not uniform. This can be explained with this example. Let's consider a σ^o measurement which is labelled with an ECMWF wind speed of 25 m/s. The true speed is more likely to be overestimated than underestimated because winds just below 25 m/s are more common than winds just above 25 m/s. If $P(u_{\mathrm{MC}}, v_{\mathrm{MC}})$ is only speed dependent and it represents the atmosphere distribution, all optimised winds will be naturally 'pulled' toward the most common wind speed in the atmosphere (around 10 m/s), thus cancelling ECMWF speed bias.

[2]This is true if errors are Gaussian in wind components.

6.2 Model inversion

6.2.1 Different techniques

The inversions of CMOD4 and CMOD-IFR2 are performed in linear space. The inversion is obtained by minimising a maximum likelihood estimator (MLE) for varying speed and direction of the simulated data (Stoffelen and Anderson, 1997a):

$$\mathrm{MLE} = \sum_{i=1}^{3} \left[\frac{\sigma^o_{o,i} - \sigma^o_{s,i}}{K_{p,i}\, \sigma^o_{o,i}} \right]^2 \tag{6.1}$$

where i runs over the 3 beams, s stands for simulated data (from the model) and o stands for observed data. $K_{p,i}$ is a dimensionless constant determined by instrumental noise (around 0.05). To determine the values of $K_{p,i}$, Stoffelen and Anderson (1997a) had to make the assumption that CMOD4 fits the data well in $\boldsymbol{\sigma^o}$ space. In fact the values of $K_{p,i}$ they find are mostly dependent on the varying distance between the surface defined by CMOD4 and the target surface in $\boldsymbol{\sigma^o}$ space. So their values must not be reused. Looking at plots such as Figure 4.5(b), one can infer that the noise is roughly constant in log space, so normalisation by $\boldsymbol{\sigma^o}$ noise level is useless in that space. Thus, the method proposed here is performed in logarithmic space, and it does not take any σ^o noise level difference between the different antennae. This procedure is a simple minimisation of (squared) distances in $\boldsymbol{\sigma^o}$ log space:

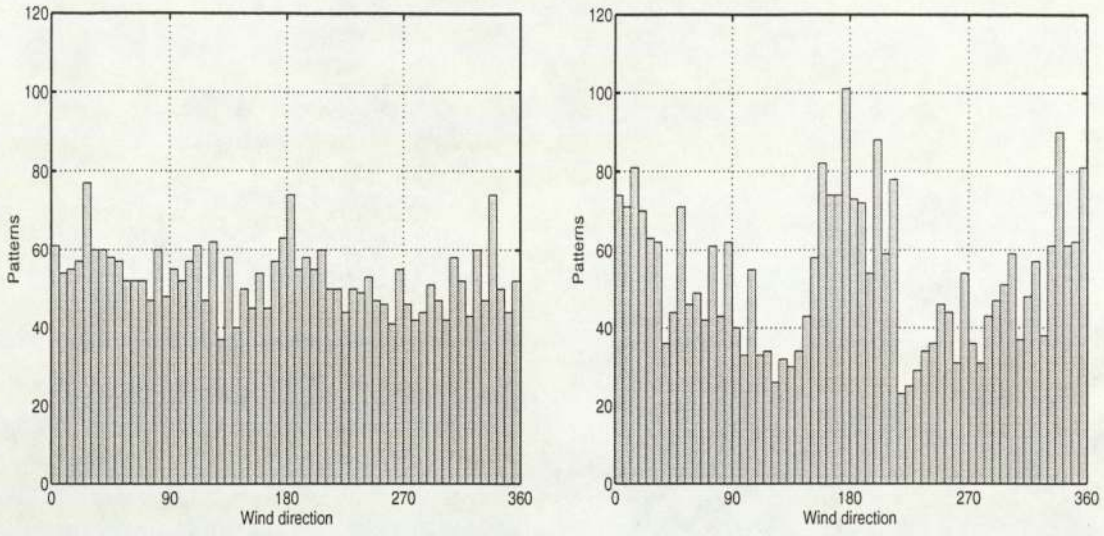
$$d = \|\boldsymbol{\sigma}^o_o - \boldsymbol{\sigma}^o_s\|^2 \tag{6.2}$$

$$= (\sigma_{f,o} - \sigma_{f,s})^2 + (\sigma_{m,o} - \sigma_{m,s})^2 + (\sigma_{a,o} - \sigma_{a,s})^2 \tag{6.3}$$

One single big lookup table is used where the simulated $\boldsymbol{\sigma^o}$ values are computed at regular intervals of direction, and at different speeds. In order to make the grid of points more regular and square on the cone, the increment in speed is adjusted every 0.5 m/s so the increment in $\|\boldsymbol{\sigma^o}\|$ is equal along the cone. The (constant) increment in direction was found empirically by making the grid look as square as possible on a plot in $\boldsymbol{\sigma^o}$ space. Instead of finding several minima and then removing ambiguity, a subsample of the lookup table was taken for $\vartheta_s = [\vartheta_{\mathrm{ECMWF}} - 90, \vartheta_{\mathrm{ECMWF}} + 90]$, thus systematically choosing the

closest solution to ECMWF background wind.

This method gave very poor results actually. As shown in Figure 6.2, the distribu-



(a) (b)

Figure 6.2: Probability density functions of ECMWF wind direction (a) and retrieved wind direction after inversion of nn2cmod (b) for track 11.

tion of the wind directions is very different for the ECMWF winds and for the retrieved winds, with marked peaks and troughs approximately 90° apart. An explanation for this phenomenon is the elliptic shape of the cross-section of the cone in σ^o space (see Figure 6.3(c)). Stoffelen and Anderson (1997a) faced the same problem and solved it by using a transformed space where the cross section was closer to a circle. Using the same method, inverted wind speeds were plotted against ECMWF wind speeds. They were in good agreement (not shown). However, both the model and the data set were biased in the same way because of the method used for data selection. Therefore such a plot cannot reveal this bias.

A complete description of the inversion method for CMOD-IFR2 is given in (Maroni *et al.*, 1995). It may be reusable for any other model developed in this thesis. The principle is recalled here for convenience. The inversion is performed in three steps:

1. For each direction between 0° and 355° with a 5° step, the speed for which MLE is minimum is found using the Newton-Raphson method.

2. Local minima in direction are then found using the 72 MLE values. The model is scanned again around these minima with a 1° precision.

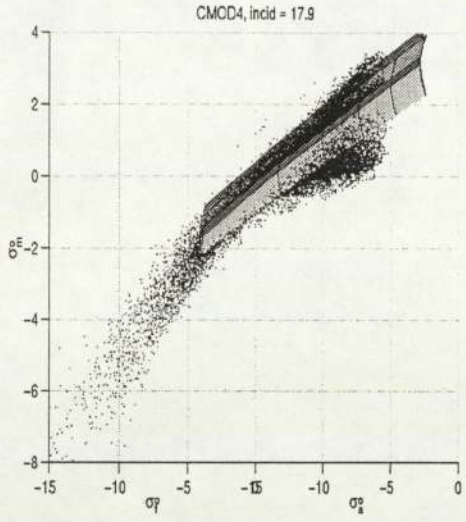
3. Ambiguity between the several selected aliases is removed.

It is interesting to note that an inversion method can be assessed by propagating a grid of winds through a model and adding Gaussian noise to the σ^o triplets obtained. These triplets can be inverted. Compared to the initial winds, the quality of the retrieved winds will mostly depend on the inversion method, not on the model.

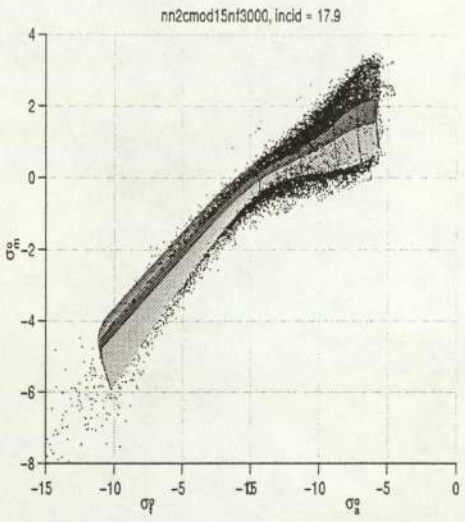
6.2.2 Potential accuracy of wind retrieval

It has been suggested in various articles that the impact of an instrumental noise of 0.2 dB in the satellite measurements is 0.5 m/s and 20° on the associated wind vectors on average. Actually, these two values depend on the track number and on the wind vector. Figures 6.4 and 6.5 show to what extent a variation of 0.2 dB can affect the retrieved winds using nn2cmod and CMOD4 respectively.
These plots were obtained using a model but no actual data. First, some reference values of σ^o are computed at regular intervals of speed, for all tracks and for a given wind direction. Then the direction is altered by a small value (10°) or the speed is altered by a small value (0.5 m/s). The corresponding σ^o are recomputed. The distance in σ^o space between the reference value and the new value is computed. The wind (or direction) which would have given rise to a change of 0.2 dB is obtained using linear interpolation.

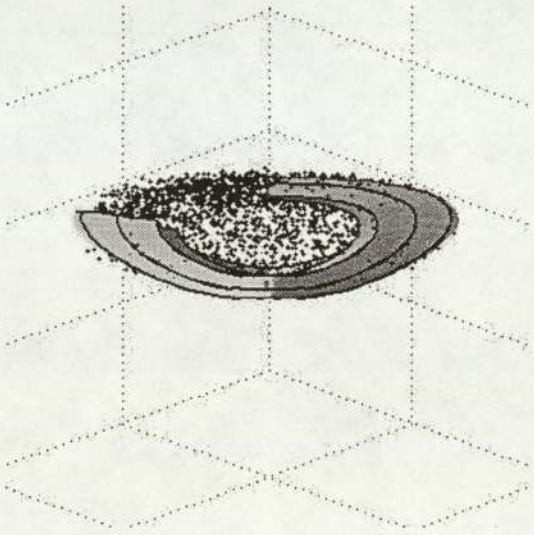
Figures 6.4 and 6.5 tell us about the effect of instrument noise on the accuracy we can can expect from an algorithm of wind vector retrieval. This depends on the model being considered. Comments below mainly rely on Figure 6.4 rather than Figure 6.5.
The best accuracy is always obtained for track 19, it decreases with the track number. For track 1, the accuracy is very poor. There are two reasons for that. First the range of σ^o values is smaller than for the other tracks (see the 'stack of funnels' page 12). Second, the spread of the points seems higher than 0.2 dB; one cannot easily distinguish the surface for σ^o measurements in σ^o space for that track. Thus, results may be improved by separating track 1 from the other tracks as its characteristics are different from them. Maybe tracks 2 and 3 should be treated independently as well. If instrumental noise is assumed to be



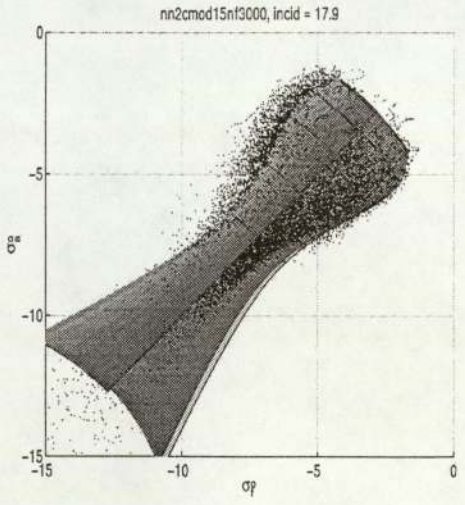
(a) The fitting of CMOD4 is poor for track 1.



(b) nn2cmod fits better than CMOD4 at high wind speeds



(c) View through the cone (nn2cmod, track 11, axes are square)



(d) Same as above, viewed from top. Note how asymmetric the distribution of the points is.

Figure 6.3: Different views in σ^o log space. The surfaces are drawn for 4-24 m/s, the lines represent constant speeds of 8, 12, 16, 20 m/s. The points are labelled with ECMWF speeds up to 25 m/s.

constant across tracks, this implies other geophysical parameters may be playing a role in track 1. Finally, the measurements are definitely not symmetric for symmetric wind directions for that track.

As far as the dependency of the error on wind vectors is concerned, the direction retrieval is best at intermediate speeds (the radius of the cone is the largest). The errors are higher at 0 and 90° than at 45 or 135°: this is due to the elliptic shape of the section of the cone. In most cases the direction error is around 5-10°, not 20°. Finally, the uncertainty on speed is highest at high wind speeds as σ^o varies slowly in these conditions (on the cone, the lines of constant speed are tight).

6.3 Possible means of improvement

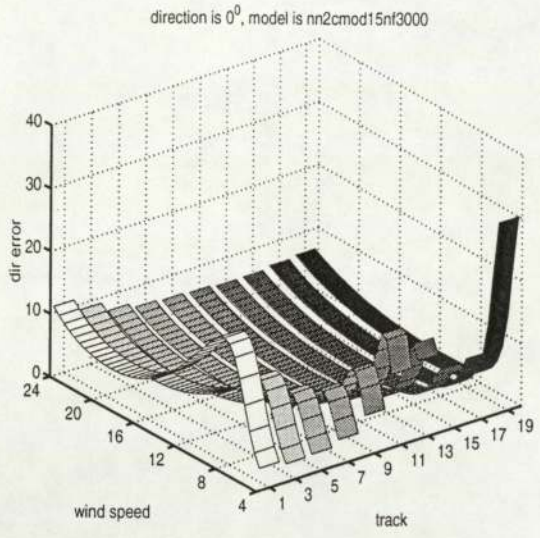
It was shown that the measurements are not symmetric for track 1. Several models where symmetry is not constrained by a functional form agree in some asymmetry for all tracks (this asymmetry can be noticed on Figure 4.5(b)). So perhaps a simple MLP network trained as in Chapter 5 (*i.e.* accounting for input noise) would learn the asymmetry and improve the results. It is not clear whether a separate model should be used for track 1 alone. In this case it would ideally take other geophysical parameters into account.

What are the ideal input parameters for the model? The dependency on the incidence angle is high at low incidence angles so $\sin(\theta)$ is a better input than $\cos(\theta)$. Using the wind Cartesian components is a very bad choice. This alone may explain the difference between the MLP and nn2cmod at high wind speeds (see cones in σ^o space, page 52). The wind vector could be represented by its speed and $\cos(\vartheta)$, $\cos(2\vartheta)$, $\sin(\vartheta)$ and maybe other harmonics.

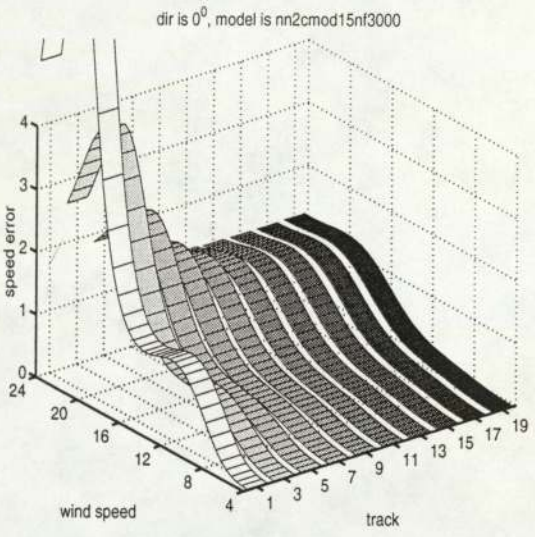
If the asymmetry is due to the satellite movement, then the comportment of the three antennae will not be the same, *i.e.* the use of a different model for each antenna will be necessary.

The Bayesian framework which was used to train nn2cmod didn't allow for the determination of hyper-parameters. The input noise level in particular is not well known. A proper Bayesian learning method could determine it.

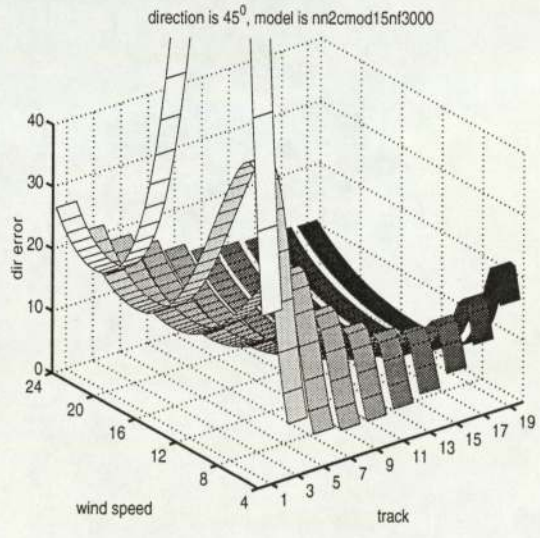
Following the advice in (Stoffelen and Anderson, 1997b), potential spatial correlation



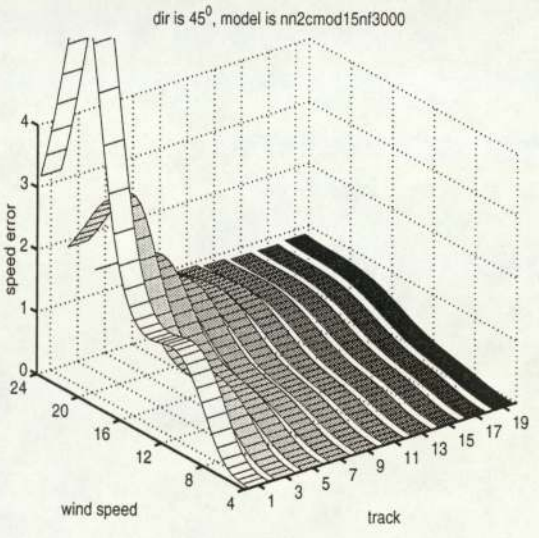
(a) Direction uncertainty for upwind



(b) Speed uncertainty for upwind

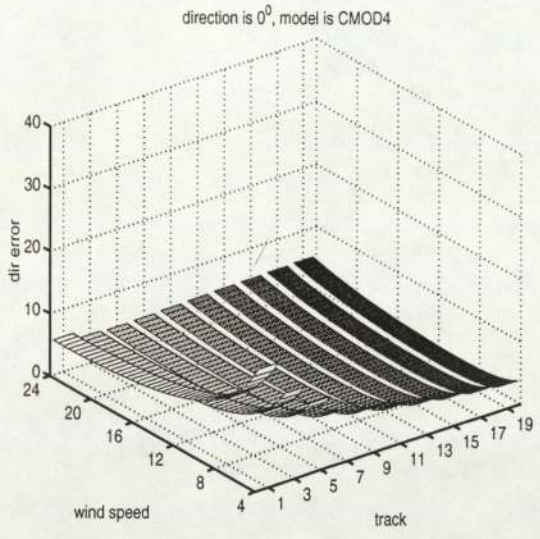


(c) Direction uncertainty for crosswind

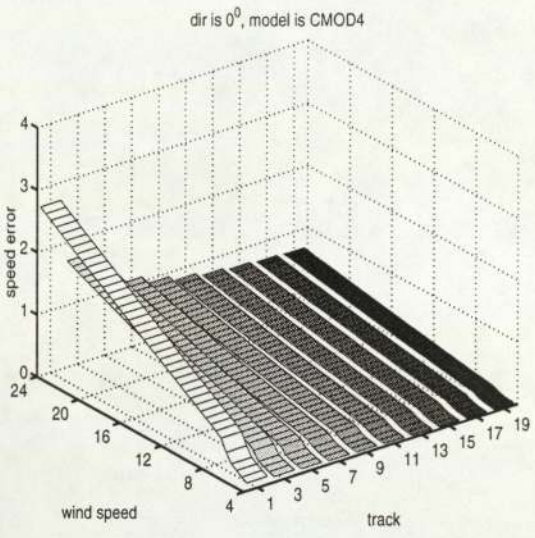


(d) Speed uncertainty for crosswind

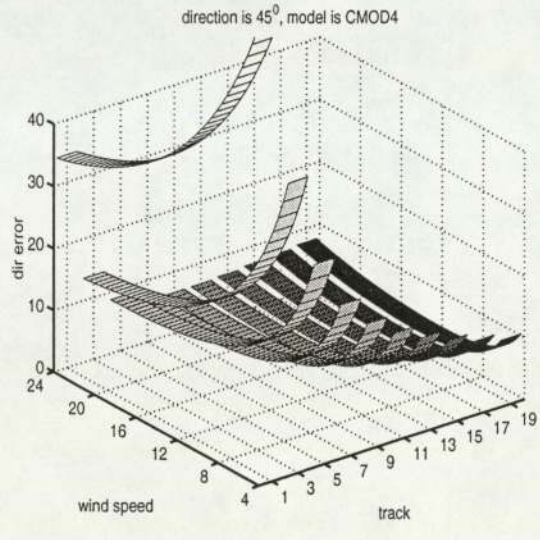
Figure 6.4: These plots show the difference in speed and direction which arise when we perform a movement of 0.2 dB (typical noise) in σ^o space along the surface defined by nn2cmod (30 h.u., 3,000 iterations). These plots are not reliable for speeds below 6-8 m/s.



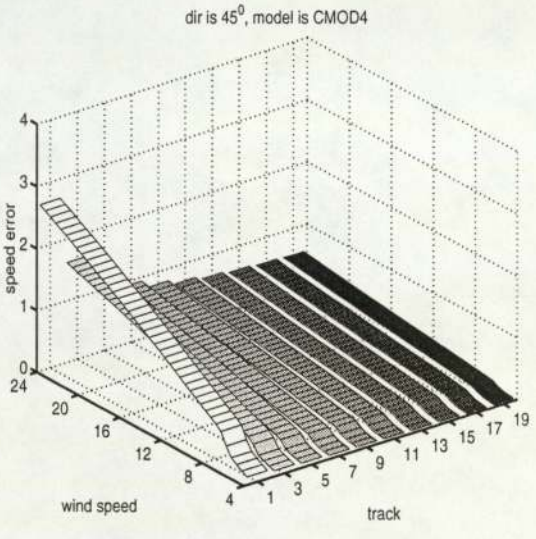
(a) Direction uncertainty for upwind



(b) Speed uncertainty for upwind



(c) Direction uncertainty for crosswind



(d) Speed uncertainty for crosswind

Figure 6.5: As on the previous page, but this is obtained using CMOD4. The σ^o dependency on speed is almost linear, which is wrong. The structure of CMOD4 is certainly not complex enough to catch the mapping accurately.

should be removed by imposing chosen cells to be more than 300 km away. In the current data, some of the σ^o triplets are aligned in σ^o space for high wind speeds, this can be seen on plots such as Figure 4.5 (there are aligned points at the top right of the plot). It means these measurements are not independent.

6.4 Conclusions

It was shown in this chapter how carefully the data selection should be done. Data must not be selected on the basis of noisy ECMWF wind speeds. There are two drawbacks in this method. The errors in the data are no longer centred so the model will learn a biased regression. And the selection of the σ^o measurements is arbitrary so there are still a lot of points which correspond to true very low wind speeds; we don't want to train our model on this data. Geometrical considerations in σ^o space allow us to select input density accurately and without introducing any bias in the data. The noisy speed distribution in the dataset can still be checked *a posteriori*. If the ECMWF winds errors are Gaussian in wind components, then the speed has to be biased but the training of the neural network can account for this bias.

An inversion method is proposed for the wind retrieval using a forward model. The quality of the inversion method is as important as the quality of the model itself. Although existing models are inverted in linear space, it is suggested that an inversion in logarithmic space will give better results. However the proposed method is poor because of the elliptic shape of the section of the surface in target space; using a transformed logarithmic space should improve the wind direction retrieval.

To improve the quality of the forward model, the constraint of symmetry with respect to opposite wind direction angles must be removed. In the same way, it is likely that a different model should be used for each antenna. Finally, a simple MLP network is still a very good candidate for the forward model if it is trained accounting for input noise and if the input parameters are chosen carefully.

Chapter 7

Conclusions

In this document, an algorithm of wind retrieval is proposed. It makes use of a forward model which is a mapping from the wind vectors to the satellite measurements. This model is built empirically with the help of neural networks, using wind vectors from a Numerical Weather Prediction model. These wind vectors are noisy. The noise comes from the imperfection of the numerical model. More uncertainty is added to these wind vectors as they need to be interpolated in time and in space so as to match the cells on the footprint of the satellite. We choose to describe these errors as Gaussian in wind components. The magnitude of the noise in target space is of order 0.2 dB.

It is shown that traditional training methods for these neural networks give suboptimal solutions because of the large uncertainty in the input wind vectors. Usual methods for testing the quality of the models are also very misleading in the presence of noisy inputs. A probabilistic model is built ignoring input noise, so the conditional distribution it provides is not correct. Actually, it only describes the effects of input noise in target space.

Some graphical representations proposed by Stoffelen and Anderson (1997a) (vertical and cross sections of the cone) are implemented. Two features of the models using neural networks are revealed:

1. They are not symmetric with respect to the satellite beam direction.

2. The fitting in σ^o space is poor.

The absence of symmetry is first attributed to the fact that the training set is finite

and imperfect. In the absence of good validation methods, the assumption of a symmetric behaviour of the backscatter with respect to the satellite beam direction seems easily justified by the geometry of the satellite. Thus a new model is built where the symmetry is constrained. It is advantageous to build a symmetric model because it requires 50% less data for the same accuracy.

At the very end of this study it is found that this assumption is actually not correct. The distribution of the σ^o measurements is asymmetric for all tracks on the satellite footprint. Therefore it is better not to constrain the model to have symmetric outputs using a functional form such as CMOD4 or CMOD-IFR2. If the movement of the satellite is responsible for this asymmetry, then a different model must be developed for each antenna. It would be useful to know the actual reason for that phenomenon. However, the symmetric model nn2cmod is found to be the most accurate during this study, so all efforts are concentrated on improving it.

The poor fitting of the forward models in target space is confirmed by the observation of the associated cones in σ^o space. This poor fitting is due to input noise in the ECMWF wind vectors. If we select σ^o triplets on the basis of the ECMWF winds which label them and if we plot these triplets, their spreading (several dB) is much bigger than the spreading due to the instrumental noise (0.2 dB) of the satellite (Figure 5.1). Therefore input noise cannot be neglected.

A Bayesian framework is adopted to account for input noise during the training procedure of the neural network. Bayesian learning can determine input and output noise levels (*i.e.* the error variances). Unfortunately, this technique is too computationally intensive, it has to be abandoned. An optimisation method is then used to find the Maximum A Posteriori of the posterior distribution of the parameters (the weights of the neural network and the wind vectors in the dataset) of the Markov chain used in Bayesian learning. The computational effort is reduced, so improved models are obtained. Unfortunately this technique leaves the difficult task of choosing the right values for the level of input and output uncertainty. Therefore Bayesian learning could still be useful, starting the Markov chain from the Maximum A Posteriori, in order to determine these hyperparameters.

Once the input noise is taken into account during the training procedure of the neural network, the fitting in σ^o space is significantly improved. It is far better than all existing

models, so the potential accuracy of the neural network is higher. The inversion of the model can now be considered. An inversion method in logarithmic space is proposed. Although this method is not computationally efficient, it shows that the inversion space needs to be rescaled so all directions are retrieved with equal probability. The inversion method is as important as the forward model itself.

The strongest winds are the most relevant to meteorological studies. Unfortunately, it is shown that the instrumental noise of the antennae of the satellite is most detrimental to the retrieval of high wind speeds. In order to to make the accuracy at high speeds as good as possible, one can select the training data to be more representative of high wind speeds. This must not be done using ECMWF wind data as it will systematically give rise to biased errors in the wind data. Instead, geometrical considerations must be used in σ^o space. In this space, planes of equation $\sigma_a^o + \sigma_f^o = 2\sigma_{\text{ref}}^o$ delimit slices of triplets which correspond to roughly constant speed. This will enable us to increase the input data density for high wind speeds without introducing any bias. The ECMWF wind speed distribution can be checked *a posteriori*. Moreover, this selection enables us not to select very low true wind speeds which we do not want to use during training.

In this study, all models trained accounting for input noise were compared using graphical representations. These models could also be compared objectively using test set likelihood. To compute test set likelihood for models trained with input noise, we can write the following integral:

$$< t \mid z >= \int tP(t \mid x, w, D')P(x \mid z)dx \tag{7.1}$$

It would also be interesting to test if the residuals are correlated. There was not sufficient time to do this during the project.

As the forward model must not be constrained to be symmetric, an MLP network (trained accounting for noise) is still a very good candidate for the task of building an empirical algorithm of wind retrieval. The inputs should be the wind speed plus several harmonics of the wind direction. For the incidence angle, $\sin(\theta)$ should be used rather than $\cos(\theta)$. Fine-tuning for the forward model will also involve improving the specification of the errors. From the two plots on page 56 (Figure 5.1), it seems the uncertainty on wind

speed is more important (the distribution is heavy-tailed) than the uncertainty on wind direction.

Although more work has to be done, there is still a lot of room for improving the current best models. Adapting the same neural networks to retrieve winds from other satellites should also be possible.

Appendix A

Symbol conventions

CERSAT	Centre ERS d'Archivage et de Traitement
ECMWF	European Centre for Medium-Range Weather Forecasts
ERS	Earth Remote-Sensing
ESA	European Space Agency
h.u.	hidden units
IFREMER	Institut Français de la Recherche pour l'Exploitation de la Mer
MCMC	Markov Chain Monte Carlo
MDN	Mixture Density Network
MLE	Maximum Likelihood Estimator
MLP	Multi Layer Perceptron
MVDN	Multi Variate Density Network
NCRG	Neural Computing Research Group (Aston University)
NWP	Numerical Weather Prediction
RBF	Radial Basis Function
RMS	Root Mean Square
SCG	Scaled Conjugate Gradient
WNF	Wind field product of CERSAT

σ_f^o	fore beam scatterometer measurement
σ_m^o	mid beam scatterometer measurement
σ_a^o	aft beam scatterometer measurement
$\boldsymbol{\sigma^o}$	$= (\sigma_f^o, \sigma_m^o, \sigma_a^o)$
σ^o	any of the 3 scatterometer measurements
(u_r, v_r) or (u, v)	relative wind vector components in m/s
$\|\|\boldsymbol{u}\|\|$ or s	wind speed in m/s
ϑ	relative wind direction in degrees
θ	incidence angle in degrees
χ	azimuth angle in degrees

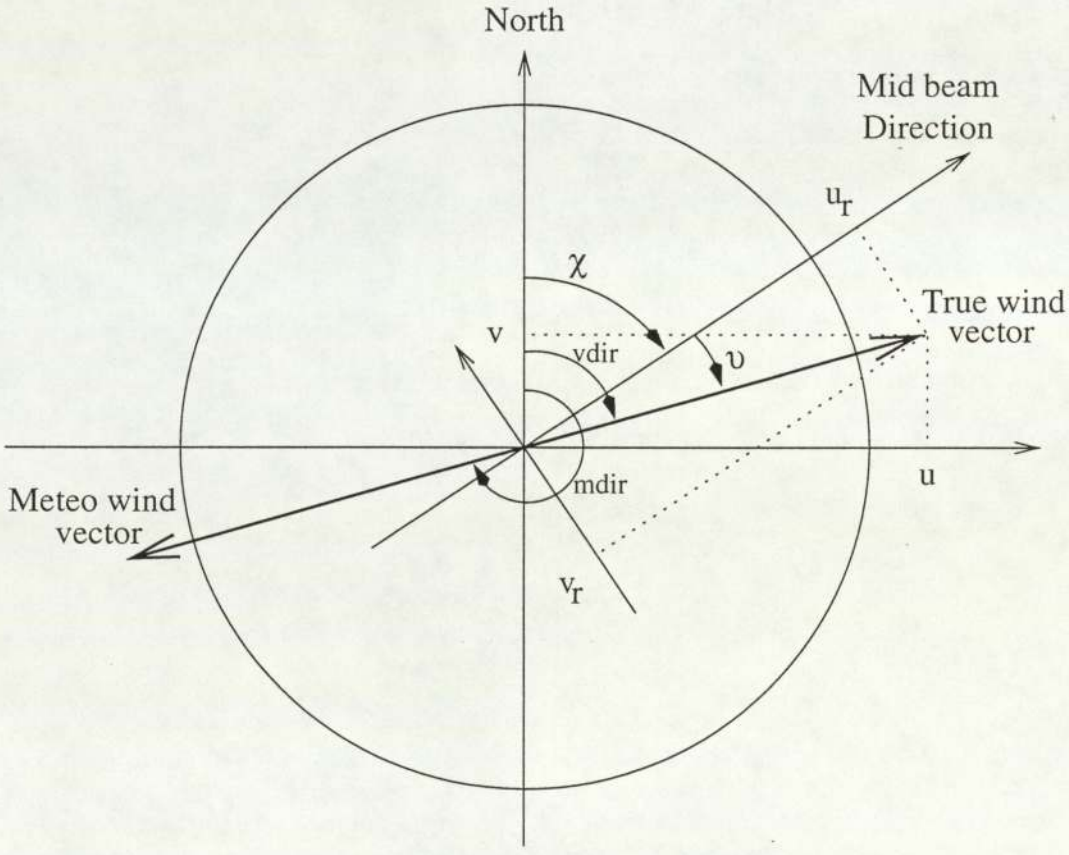


Figure A.1: Angles in satellite geometry

The wind direction is given in standard meteo : clockwise relative to north. By convention, the following notation is adopted:

- *mdir* is the direction of the standard meteo wind vector (direction of 0° is blowing from north).
- *vdir* is the direction of the true wind vector (direction of 0° is blowing from south). $vdir = (mdir + 180)_{360}$
- ϑ is the direction of the true wind vector in the coordinate system relative to the azimuth angle. $\vartheta = (vdir - \chi)_{360}$

The true wind vector is decomposed into its Cartesian components:

- in the absolute coordinate system (relative to north), the notation for the components of the true wind vector is (u, v).

$$u = \|\boldsymbol{u}\| * \cos(\frac{\pi}{2} - vdir_{\text{rad}}) \tag{A.1}$$

$$v = \|\boldsymbol{u}\| * \sin(\frac{\pi}{2} - vdir_{\text{rad}}) \tag{A.2}$$

- in the relative coordinate system (relative to the azimuth angle), the notation for the components of the true wind vector is (u_r, v_r) (or (u, v)).

$$u_r = \|\boldsymbol{u}\| * \cos(\frac{\pi}{2} - \vartheta_{\text{rad}}) \tag{A.3}$$

$$v_r = \|\boldsymbol{u}\| * \sin(\frac{\pi}{2} - \vartheta_{\text{rad}}) \tag{A.4}$$

Appendix B

Technical plots

B.1 Ellipses on the cone sections

This appendix presents the technique which was used to plot ellipses on the vertical and cross sections of the cone for the MVDN model (figures on pages 47 and 47).

The multivariate density network which is used in this study is a model for $P(\boldsymbol{\sigma}^o|u, v, \theta)$. This probability is assumed to be a Gaussian distribution with a full covariance matrix in $\boldsymbol{\sigma}^o$ space. Hence a surface of constant probability is an ellipsoid. Cutting this ellipsoid by a plane which passes through its centre results in an ellipse. We need to find the Cartesian components of the points of this ellipse in the cutting plane.

One approach is to compute the equation of the surface of the ellipsoid using the eigenvalues and eigenvectors of the covariance matrix. This equation, coupled with the equation of the plane gives the equation of the intersection, but this equation is difficult to solve.

A better approach is to compute, for a set of directions $\alpha_i \in [0, 2\pi]$ around the centre, in the cutting plane, the standard deviation of the Gaussian distribution along each direction α_i. It is possible to compute this standard deviation because the distribution is a one-dimensional Gaussian along a straight line passing through the centre. This standard deviation gives the radius of the ellipse along the considered direction. Here are more details.

Let $\mathcal{B}$ be the basis of the space defined by the axes of the cutting plane and its normal

direction. The normal direction is chosen so that $\det(\mathcal{B}) = 1$. For the vertical section,

$$\mathcal{B} = \begin{pmatrix} -1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 \\ -1/\sqrt{2} & 0 & -1/\sqrt{2} \end{pmatrix} \tag{B.1}$$

and for the cross section:

$$\mathcal{B} = \begin{pmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \end{pmatrix}. \tag{B.2}$$

We also need the first vector of the basis $\mathcal{B}$ which will be rotated to obtain all the directions around the centre. For the vertical and the cross section, these are respectively

$$x = \begin{pmatrix} -1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{pmatrix} \tag{B.3}$$

Then for a set of values $\alpha_i \in [0, 2\pi]$, we define a rotation matrix of angle α_i around the third axis:

$$\mathcal{R}_1 = \begin{pmatrix} \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{B.4}$$

Using $\mathcal{B}$ as a transformation matrix, then the rotation matrix around the normal direction to the cutting plane can be written as:

$$\mathcal{R}_2 = \mathcal{B}\,\mathcal{R}_1\,\mathcal{B}^{-1} \tag{B.5}$$

So now, for each $\alpha_i \in [0, 2\pi]$, we can consider each direction:

$$x_i = \mathcal{R}_2\, x \tag{B.6}$$

The standard deviation along direction x_i is

$$\sigma_i = \sqrt{x_i^T \, \Sigma^{-1} \, x_i} \tag{B.7}$$

where Σ is the covariance matrix of the Gaussian distribution. (σ_i, α_i) are the polar coordinates of the points of the ellipse with respect to its centre. The transformation to Cartesian components with respect to the origin is easily obtained.

B.2 3D Cone

This section presents the MATLAB source code for displaying a cloud of points in the σ^o space together with the surface defined by a given forward model passed as argument to the function. The second argument of the function is a matrix with 3 columns of $(\sigma_f^o, \sigma_m^o, \sigma_a^o)$ log values, corresponding to a single mid beam incidence angle. This angle is the 3rd argument of the function.

In the code, `phi` is the relative wind direction ϑ in radians.

A call to the function `modelfwd` can be found in the code. It takes the name of a model as argument, with the inputs (u_r, v_r) and θ and returns the σ^o triplet. The latter function was found to be extremely useful as it makes transparent all specificities of the different models (pre-processing, single or triple output models...). However, the implementation of this function is very specific so it is not presented here.

```
function disp_cone(model,trace5,incid)
%   disp_cone(model, trace5)        % assumes incid = 34.9
%   disp_cone(model, trace9, 45.5)
%   disp_cone(model, trace0, 17.9)
%   disp_cone(model, trace1, 21.5)

%speed_ranges = [6 6; 9 10; 13 13];
speed_ranges =[4 8; 8 12; 12 16; 16 20; 20 24]

%phi_ranges = [0 2*pi];                    % full cone
%phi_ranges = [0 pi/2; pi pi+pi/2];        % remove a half
phi_ranges = [0 3*pi/4; pi pi+3*pi/4];     % remove a quarter

%phi_ranges = phi_ranges+pi/4;             % rotate removed part 1/4 turn
%phi_ranges = phi_ranges+pi/2;             % rotate removed part 1/2 turn
```

```
% one figure per incidence angle
if nargin < 3
   incid = [34.9];  %trace5
   %incid = [18.0 21.7 25.2 28.6 31.8 34.9 37.7 40.5 43.0 45.4];
end

% ----------------------------------------

% number of speed and phi ranges (one piece of surface each)
n_speed_ranges = size(speed_ranges, 1);
n_phi_ranges   = size(phi_ranges, 1);

for i=1:length(incid)

   fh = figure;
   for i_speed_range = 1:n_speed_ranges

      minspeed = speed_ranges(i_speed_range,1);
      maxspeed = speed_ranges(i_speed_range,2);
      speed = (minspeed:0.5:maxspeed)';
      nspeed = length(speed);

      for i_phi_range = 1:n_phi_ranges

         minphi = phi_ranges(i_phi_range,1);
         maxphi = phi_ranges(i_phi_range,2);
         phi  = minphi:pi/60:maxphi;
         nphi = length(phi);

         ur = speed*cos(pi/2-phi);
         vr = speed*sin(pi/2-phi);

         ur = reshape(ur,nphi*nspeed,1);
         vr = reshape(vr,nphi*nspeed,1);
         theta = incid(i)*ones(size(ur));

         sig = modelfwd(ur, vr, theta, model);

         % reshape outputs
         % msig stands for matrix sig
         msig1 = reshape(sig(:,1), nspeed, nphi);
         msig2 = reshape(sig(:,2), nspeed, nphi);
         msig3 = reshape(sig(:,3), nspeed, nphi);

         % Note the re-ordering of msig:
         % mid beam <=> z axis
```

```
        if min(size(msig1)~=1)
           surf(msig1, msig3, msig2, ones(size(speed))*phi);
        end
        hold on

        % plot lines of constant extreme speeds in back
        plot3(msig1(end,:),msig3(end,:),msig2(end,:), ...
             '-k', 'Linewidth',2);
        plot3(msig1(1,:),msig3(1,:),msig2(1,:), '-k', ...
             'Linewidth',2);

        % plot lines of constant theta in blue,
        % unless they superimpose (no section)
        if mod(minphi,2*pi) ~= mod(maxphi,2*pi)
           plot3(msig1(:,1),msig3(:,1),msig2(:,1), '-b', ...
                'Linewidth',2);
           plot3(msig1(:,end),msig3(:,end),msig2(:,end), ...
                '-b', 'Linewidth',2);
        end
     end
  end

  figname = [model, ', incid = ', num2str(incid(i))];
  title(figname)
  xlabel('\sigma^o_f')
  ylabel('\sigma^o_a')
  zlabel('\sigma^o_m')
  shading interp
  c1 = [ ones(16,1), linspace(0,1,16)', zeros(16,1) ];
  c2 = [ linspace(1,0,16)', ones(16,1), zeros(16,1) ];
  c3 = [ linspace(0,1,16)', linspace(1,0,16)', linspace(0,1,16)'];
  c4 = [ ones(16,1), zeros(16,1), linspace(1,0,16)'];
  colormap([c1; c2; c3; c4]);
  % colormap([autumn; flipud(autumn)]);
  caxis([0 2*pi]);
  view(45,15)
  set(fh,'Position',[422 227 818 749]);
  set(fh,'Name',[ 'Cone: ', model]);
end

disp('from aside       view(45,0)')
disp('from front       view(135,34)')
disp('aerial view      view(100,35)')
disp('from top         view(0,90)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot cloud of points
```

```
% remove left half of the points
rmhalf = 0;

% select points for a range of ECMWF speeds
show_noise = 0;

% values > 1 reduce the number of points
step = 1;

% ----------------
if 1 & not(length(incid)>1)    % 0: do not plot the points

   if nargin<2
      load trace5.dat;
   end
   data = trace5;

   if rmhalf
      siga  = data(:,3);
      sigf  = data(:,1);
      ind = find(sigf>siga);
   else
      ind = 1:length(data);
   end

   if show_noise ==1
      % select speed range
      datspeed = data(:,6);
      ind = find(20<datspeed);
   elseif show_noise ==2
      datdir = data(:,7);
      dataz = data(:,5);
      datspeed = data(:,6);
      [nil1 nil2 datrdir] = speed_mdir_to_uv(datspeed, ...
           datdir, dataz);
      datrdir = datrdir * pi/180;
      ind = find(datrdir>(pi/4-pi/20) & datrdir<(pi/4+pi/20));
   end

   sigf  = data(ind,1);
   sigm  = data(ind,2);
   siga  = data(ind,3);

   plot3(sigf(1:step:end), siga(1:step:end), sigm(1:step:end),...
        '.k','markersize',4);

end
```

Appendix C

Training with noise: error derivatives

The following derivatives are given in this section:

$$\frac{\partial E_i}{\partial \vartheta}, \quad \frac{\partial E_i}{\partial s}, \quad \frac{\partial E_i}{\partial w}, \qquad i = 1...3 \tag{C.1}$$

The expressions for the errors E_i are given in Section 5.5.2 (page 64). Figure 3.1 (page 38) is very helpful to understand these calculations.

For simplicity, the wind speed $\|\boldsymbol{u}\|$ will be noted s in this section. A tilde upon a symbol denotes a normalised value. The normalisations are done with respect to the mean and variance of the training set for each variable. These variances are noted σ_ϑ^2 and σ_s^2, they do not correspond to error variances.

$$\tilde{s} = \frac{s - \bar{s}}{\sigma_s}, \qquad \tilde{\vartheta} = \frac{\vartheta - \bar{\vartheta}}{\sigma_\vartheta} \tag{C.2}$$

Error E_1

This is the tricky bit. The relationship between E_1 and the inputs of the model is obtained through the whole architecture shown on Figure 3.1.

$$E_1 = \frac{\sum(\sigma^o - \sigma_t^o)^2}{2\sigma_{\sigma^o}} \tag{C.3}$$

And σ^o is given by the nn2cmod model:

$$\sigma^o = L\,(B_0 + p \log K) \qquad \text{with} \qquad L = \frac{\log 10}{10} \tag{C.4}$$

$$\text{and} \qquad K = (0.1 \tanh B_1 \, \cos(\vartheta_{\text{rad}}) + 0.8 \tanh B_2 \, \cos(2\vartheta_{\text{rad}})) \tag{C.5}$$

- $\partial E_1 / \partial w$ has already been calculated in section 3.2, using back-propagation.

- $\partial E_1 / \partial \vartheta_{\text{deg}}$: Here the units for ϑ will be specified explicitly: ϑ_{rad} (used in trigonometry) or ϑ_{deg} (the input). Keeping the notations for K and L, we can compute $\partial E_1 / \partial \vartheta_{\text{deg}}$ rather easily as we do not have to derive through the network:

$$\frac{\partial E_1}{\partial \vartheta_{\text{deg}}} = L\,p\,\frac{\pi}{180}\,\frac{(-0.1 \tanh B_1 \, \sin(\vartheta_{\text{rad}}) - 0.8 * 2 \tanh B_2 \, \sin(2\vartheta_{\text{rad}}))}{K}\,\frac{\sigma^o - \sigma_t^o}{2\sigma_{\sigma^o}} \tag{C.6}$$

- Deriving E_1 with respect to speed is more complicated as the dependency upon speed is obtained through the network. We first calculate the derivatives of the outputs of the network with respect to the inputs in a general context. Using the same notation as in (Bishop, 1995), the outputs of a MLP are related to the inputs by:

$$y_k = a_k = \sum_{j=1}^{M} w_{kj}\, g\Big(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\Big) + w_{k0} \tag{C.7}$$

There are two inputs in our model:

$$y_k = \sum_{j=1}^{M} w_{kj}\, g\Big(w_{j1}\tilde{\theta} + w_{j2}\tilde{s} + w_{j0}\Big) + w_{k0} \tag{C.8}$$

g was chosen as the tanh function, with linear output units so $y = a$. As the normalised speed $\tilde{s} = (s - \bar{s})/\sigma_s$, the derivatives of the outputs with respect to the speed are given by:

$$\frac{\partial a_k}{\partial s} = \frac{1}{\sigma_s} \sum_{j=1}^{M} w_{kj} w_{j2} \;/\; \cosh^2\Big(w_{j1}\tilde{\theta} + w_{j2}\tilde{s} + w_{j0}\Big) \tag{C.9}$$

The $y_k, k = 1...4$ are the 4 outputs B_1, B_2, B_3, p. So the derivatives above are denoted B'_0, B'_1, B'_2, p'. Now we can write:

$$\frac{\partial E_1}{\partial s} = L\left(B'_0 + p' \log K + p\frac{K'}{K}\right)/2\sigma_{\sigma^o} \tag{C.10}$$

where

$$K' = \frac{\partial K}{\partial s} = \frac{0.1\cos(\vartheta_{\text{rad}})\, B'_1}{\cosh^2 B_1} + \frac{0.8\cos(2\vartheta_{\text{rad}})\; B'_2}{\cosh^2 B_2} \tag{C.11}$$

Error E_2

This error is associated with the the distance between (u_t, v_t) vectors from the training set and computed (u, v) vectors. If we note $\Delta U_n = \big((u_n - u_{n,t}),\ (v_n - v_{n,t})\big)^T$, then E_2 can be written as:

$$E_2 = \frac{1}{2}\sum_1^N (\Delta U)^T\, \Sigma_{u,v}^{-1}\, (\Delta U) \tag{C.12}$$

where

$$\Sigma_{u,v} = \begin{pmatrix} \sigma_{u,v} & 0 \\ 0 & \sigma_{u,v} \end{pmatrix} \tag{C.13}$$

and $\sigma_{u,v}$ is the estimation of the standard deviation of the errors in (u, v) space for NWP winds. Consequently, for each pattern (indices n are removed):

$$\frac{\partial E_2}{\partial \vartheta_{\text{deg}}} = (\Delta U)^T\, \Sigma_{u,v}^{-1}\, \frac{\partial(\Delta U)}{\partial \vartheta_{\text{deg}}} \qquad \text{and} \qquad \frac{\partial E_2}{\partial s} = (\Delta U)^T\, \Sigma_{u,v}^{-1}\, \frac{\partial(\Delta U)}{\partial s} \tag{C.14}$$

ΔU can be written in polar coordinates. Then the derivatives with respect to speed and direction are:

$$\frac{\partial(\Delta U)}{\partial \vartheta_{\text{deg}}} = \frac{\pi}{180}\begin{pmatrix} s\,\cos(\vartheta_{\text{rad}}) \\ -s\,\sin(\vartheta_{\text{rad}}) \end{pmatrix} \qquad \text{and} \qquad \frac{\partial(\Delta U)}{\partial s} = \begin{pmatrix} \sin(\vartheta_{\text{rad}}) \\ \cos(\vartheta_{\text{rad}}) \end{pmatrix} \tag{C.15}$$

Finally, E_2 does not depend on the network weights so $\partial E_2/\partial w = 0$.

Error E_3

This error depends only on speed, so

$$\frac{\partial E_3}{\partial w} = 0 \qquad \text{and} \qquad \frac{\partial E_3}{\partial \vartheta_{\text{deg}}} = 0 \tag{C.16}$$

Above 28 m/s $\partial E_3/\partial s = (s - 28)/\delta s$ and below 4 m/s $\partial E_3/\partial s = (s - 4)/\delta s$ where δs is the uncertainty of NWP wind speeds.

Implementation details

For the implementation it is important to give a vectorized expression of all equations above in order to keep training times reasonable. For instance Equation C.9 is not trivial to obtain: in MATLAB code, this equation can be written as follows for the derivative of the 1st output with respect to the 2nd input $\partial B_0/\partial \tilde{s}$ (a similar expression gives the derivatives of the 3 other outputs):

```
(1./ cosh(x*net.w1 + repmat(net.b1, nx, 1)).^2)
   * (net.w2(:,1).*net.w1(2,:)')
```

where `x` is 10,000*2 (10,000 input patterns), `net.w1` is 2*10 (input weights, for 10 hidden units), `net.b1` is 1*10 (input biases), `net.w2` is 10*4 (output weights), `net.b2` is 1*4 (output biases), `nx` = 10,000. The biggest matrix resulting from that computation is 10,000*10 (patterns*hidden units), which is very affordable.

Appendix D

The nn2cmod transfer function

The functional form of this model is

$$\sigma_m^o = \frac{10}{\ln(10)}\Big(b_0 + p\ln\left(1 + 0.1\tanh(b_1)\cos(\vartheta) + 0.8\tanh(b_2)\cos(2\vartheta)\right)\Big)$$

B_1, B_2, B_3 and p (in this order) are determined by the four outputs of a neural network with two inputs and 15 hidden units:

$$y_k = \sum_{j=1}^{15} w_{kj}^{(2)} \tanh\left(w_{j0}^{(1)} + w_{j1}^{(1)}\tilde{\theta} + w_{j2}^{(1)}\tilde{s}\right) + w_{k0}^{(2)} \qquad k = 1\ldots 4$$

$$\text{where} \qquad \tilde{\theta} = \frac{\sin(\theta) - 0.6190}{0.1463} \qquad \text{and} \qquad \tilde{s} = \frac{s - 12.3534}{5.9767}$$

The weights of the network are presented on the next page.. The input and output biases ($w_{:,0}^{(1)}$ and $w_{:,0}^{(2)}$) are at the first line of each matrix.

$$w^{(1)} = \begin{pmatrix} 0.067894 & -0.30723 & -1.2241 \\ -1.1824 & -0.24107 & -0.87225 \\ 2.1596 & -0.35863 & 0.8856 \\ -1.58 & 0.19281 & 0.57725 \\ -0.28322 & -0.301 & -0.16 \\ -4.0252 & -0.087717 & -2.4489 \\ -2.7307 & 2.5937 & 0.025007 \\ 2.1209 & 0.65365 & 0.49967 \\ 0.086781 & 0.49556 & 0.045686 \\ -0.59721 & 0.42012 & 0.43697 \\ 0.86032 & 0.18014 & -0.034548 \\ -1.129 & -0.64124 & -0.074035 \\ -1.7694 & 0.93374 & -0.059592 \\ 1.0992 & -0.12872 & -0.23182 \\ -0.55728 & 0.6679 & 0.37155 \end{pmatrix}$$

$$w^{(2)} = \begin{pmatrix} -2.2188 & -0.15465 & -0.35361 & 1.9599 \\ -0.37344 & -0.14064 & 0.017999 & -0.52902 \\ -1.29 & -0.40896 & -0.20945 & 0.84328 \\ 0.81448 & 0.43343 & 1.4056 & -1.4742 \\ 0.90041 & 0.26926 & -0.0065368 & -0.36981 \\ 1.2265 & 1.4669 & -0.53272 & -0.84362 \\ -2.0237 & 0.081579 & 0.79211 & 0.19508 \\ -0.2417 & 0.37919 & 0.20758 & -0.22146 \\ -2.0705 & 0.064967 & -0.45171 & 1.5472 \\ -1.2798 & -0.65595 & -0.095132 & 0.74194 \\ 0.88336 & 0.74858 & -0.77331 & -0.82517 \\ 0.18704 & 0.08397 & 0.34015 & -0.72164 \\ 1.4216 & 1.2515 & 0.0030994 & -0.1294 \\ 0.62342 & -0.20165 & -0.4667 & 0.45718 \\ 0.42097 & -0.022376 & -0.3991 & -0.17022 \\ -0.21451 & -0.22747 & 0.49072 & -0.64858 \end{pmatrix}$$

References

Bishop, C. M. 1994, February. Mixture Density Networks. Technical Report NCRG/94/004, Neural Computing Research Group.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford Univ. Press.

Cornford, D. and I. T. Nabney 1998a, February. Cost Functions. Technical report, Neural Computing Research Group. Unpublished.

Cornford, D. and I. T. Nabney 1998b, May. NEUROSAT: An Overview. Technical Report NCRG/98/011, Neural Computing Research Group.

Cornford, D., I. T. Nabney, and C. M. Bishop 1997. Neural Network Based Wind Vector Retrieval from Satellite Scatterometer Data. Technical report, Neural Computing Research Group.

Evans, D. J. 1998. Mixture Density Network Training by Computation in Parameter Space. Technical Report NCRG/98/016, Department of Computer Science and Applied Mathematics, Aston University, Birmingham, B4 7ET, UK.

Janssen, P. A. E. M., H. Wallbrink, C. J. Calkoen, D. V. Halsema, W. A. Oost, and P. Snoeij 1998, April. VIERS-1 scatterometer model. *Journal of Geophysical Research* **103** (C4), 7807–7831.

Maroni, C., N. Grima, and Y. Quilfen 1995, April. Quality Assessment of CERSAT Wind Fields Products. Technical report, CERSAT.

Mejia, C., F. Badran, A. Bentamy, M. Crepon, S. Thiria, and N. Tran 1998. Determination of the geophysical model function of NSCAT scatterometer by the use of neural networks. Technical report, LODYC, Universite de Paris 6.

Mejia, C., S. Thiria, N. Tran, M. Crepon, and F. Badran 1998, June. Determination of the geophysical model function of the ERS-1 scatterometer by the use of neural networks. *Journal of Geophysical Research* **103** (C6), 12853–12866.

Nabney, I. T. and C. M. Bishop 1995a. Modelling conditional probability distributions for periodic variables. In *4th International Conference on Artificial Neural Networks*, pp. 177–182. IEE.

Nabney, I. T. and C. M. Bishop 1995b. Modelling wind direction from satellite scatterometer data. In *International Conference on Artificial Neural Networks*.

Neal, R. M. 1996. *Bayesian Learning for Neural Networks*. Springer. Lecture Notes in Statistics 118.

Offiler, D. 1994. The Calibration of ERS-1 Satellite Scatterometer Winds. *Journal of Atmospheric and Oceanic Technology* **11**, 1002–1017.

Rufenach, C. 1995. A new relationship between radar cross-section and ocean surface wind speed using ERS-1 scatterometer and buoy measurements. *Int. J. Remote Sensing* **16** (18), 3629–3647.

Rufenach, C. 1997. Comparison of Four ERS-1 Scatterometer Wind Retrieval Algorithms with Buoy Measurements. *Journal of Atmospheric and Oceanic Technology* **15**, 304–313.

Stoffelen, A. 1998, April. Toward the true near-surface wind speed: error modelling and calibration using triple collocation. *Journal of Geophysical Research* **103** (C4), 7755–7766.

Stoffelen, A. and D. Anderson 1995, February. The ECMWF contribution to the characterisation, interpretation, calibration and validation of ERS-1 scatterometer backscatter Measurements and winds, and their use in numerical weather prediction models. ESA contract report, European Centre for Medium-Range Weather Forecasts.

Stoffelen, A. and D. Anderson 1997b. Scatterometer Data Interpretation: Estimation and Validation of the transfer Function CMOD4. *Journal of Geophysical Research* **102**, 5767–5780.

Stoffelen, A. and D. Anderson 1997a. Scatterometer Data Interpretation: Measurement Space and Inversion. *Journal of Atmospheric and Oceanic Technology* **14**, 1298–1313.

Thiria, S., C. Mejia, F. Badran, and M. Crepon 1993. A Neural Network Approach for Modelling Nonlinear Transfer Functions: Application for Wind Retrieval From Spaceborne Scatterometer Data. *Journal of Geophysical Research* **98**, 22,827–22,841.

Williams, P. M. 1996. Using Neural Networks to Model Conditional Multivariate Densities. *Neural Computation* **8**, 843–854.

Wright, W. A. 1998. Neural Network Regression With Input Uncertainty. In *Neural Networks for signal processing*, Volume 8, pp. 284–293.