A fast forward model for the assimilation of radiances from the EOS-MLS

Donald Scorgie

Department of Atmospheric Science
University of Edinburgh

Thesis submitted for the Degree of Doctor of Philosophy in the University of Edinburgh

· 2006 ·

Abstract

In this thesis the idea of using neural networks as a forward model for the EOS-MLS (Earth Observation System - Microwave Limb Sounder) is considered for a direct assimilation scheme. Neural networks are a type of non-linear regression technique that can provide fast, accurate results and are used extensively in many different fields.

Here a neural network is constructed to act as a forward model for the EOS-MLS. The neural network uses a temperature profile and tangent pressure levels as inputs and produces the corresponding radiance profile for one channel of the EOS-MLS. The work here primarily concentrates on one band of the EOS-MLS that is centred on an oxygen line and whose radiances are affected only by temperature for the majority of the channels. It shows that a neural network can function as a forward model in this case, producing radiances that are within instrument noise and for most channels, within half the instrument noise.

Adding ozone to the forward model affects the radiances in only two channels of this band, increasing the radiances in some minor frames by around $\sim 10K$. It was found that this difference could be accounted for in the neural network forward model by adding ozone to the inputs. A second band, which is centred on an ozone line, is briefly considered. It was found that above $150hPa$ the radiances from this band could be modelled well using a neural network. Below this height, the neural network produced large errors in radiance (of around $1.5K$ - four times the instrument noise). This is thought to be due to the effects of water vapour.

A problem specific to limb sounders that must be faced when doing direct assimilation is determining the tangent pressures of the radiances. During retrieval, these tangent pressures are normally retrieved as part of the state vector and discarded. For an assimilation process, these tangent pressures may be unavailable and have to be deduced in some way. Here, a neural network is used to retrieve tangent pressures outside the assimilation process. These retrieved tangent pressures can then be used by the forward model and assumed to be correct. It was found that tangent pressures could be retrieved with an accuracy of around $50m$, much better than required for a forward model.

The final problem faced within this thesis is the creation of the Jacobian of the instrument forward model. This is the derivative of the radiances with respect to the state vector and is used by the assimilation process to update the model fields during the assimilation process. Traditional forward models can be differentiated automatically within code. However for neural networks this presents some difficulties. In this thesis, the neural network is differentiated analytically and the result is implemented in the code. It was found that the Jacobian for temperature can be generated which is good for much of the atmosphere but at

specific heights contains large discrepancies. It is shown that using a reduced neural network to calculate specific minor frames reduces these errors. The Jacobian for both ozone and water vapour were generated for the ozone band modelled. It was found that below $0.5hPa$, the ozone derivative was in general agreement with the true derivative but above this the derivative is much smaller than the truth. For the water vapour profile it was found that, although the general shape of the derivative is correct around the main feature, outside this the derivative deviated significantly from the true derivative.

Overall, it is shown that using a neural network forward model is a promising approach to assimilating radiances from the EOS-MLS. The neural network is significantly faster to run than a traditional forward model, while still providing good accuracy. There are several possible ways to improve the results found here. The training data used in this thesis were generated using a non-tomographic model. This will affect the accuracy of the radiances generated by around $\approx 1K$. In order to assimilate the ozone radiances, either the lower minor frames must be ignored or an approach to deal with water vapour must be found.

Contents

Acknowledgements xi

1 Introduction 1

2 Background 6

2.1 Introduction 6

2.2 Data Assimilation 6

2.2.1 Fundamental Concepts 7

2.2.2 4-Dimensional Assimilation 9

2.2.3 What is Needed for a 4D-VAR Assimilation Scheme 11

2.3 The EOS-MLS 11

2.3.1 Instrument Details 12

2.3.2 Measurements 19

2.4 The Radiative Transfer Equation 19

2.5 Neural Networks 22

2.5.1 Background 23

2.5.2 Definitions 24

2.5.3 ADALINE and MADALINE 26

2.5.4 Perceptrons 29

2.5.5 Back-propagation 33

2.6 Previous Work 38

3 Preliminary Evaluation of the Neural Network Forward Model 41

3.1 Introduction 41

3.2 The First Model 41

3.3 The Network Architecture 42

3.4 Training the Network 46

3.4.1 The Training Set 47

3.5 Results 50

3.5.1 Initial Trials 50

3.5.2 The Effect of the Number of Hidden Nodes 56

3.5.3 Improving the Initial Results 60

3.6 Discussion 63

4 Tangent Pressures 65

4.1 Introduction 65

4.2 What are Tangent Pressures? 65

4.3 The hydrostatic equation 68

4.4 Possible Solutions to the Tangent Pressure Problem 69

4.4.1 Invariant Tangent Pressures 69

4.4.2 Using Geometric Height Information 70

4.5 Acquiring Tangent Pressure Information 70

4.5.1 Traditional Retrieval 73

4.5.2 Neural Network Retrieval 73

4.6 Dealing with Noisy Radiances 78

4.6.1 Training Using Clean Radiances 79

4.6.2 Training With Noisy Radiances 82

4.7 Training a network with tangent pressure levels 84
4.8 Discussion 87

5 Extending the Neural Network **89**

5.1 Introduction 89
5.2 Extending the Network to More Channels 89
5.2.1 The Network Architecture 90
5.2.2 Training 90
5.2.3 Badly Trained Channels 92
5.2.4 The Neural Network With the Updated Training Sets . . . 96
5.3 Dealing With Chemical Species 102
5.3.1 Band One 102
5.3.2 Band Seven 107
5.4 Discussion 111

6 The Adjoint Model **117**

6.1 Introduction 117
6.2 The Adjoint Model 117
6.3 Calculating the Jacobian 120
6.4 Results 121
6.5 Tangent Pressure Jacobian 129
6.6 Jacobian for Constituant Species 136
6.6.1 Improving the Neural Network Performance 144
6.7 Discussion 144

7 Conclusions and Discussions **148**

A Further Discussion of Neural Networks **153**

A.1 Simple Perceptron 153

A.2 Multi-layered Perceptrons 155

B Definitions **160**

References **164**

C Papers **169**

List of Figures

2.1 Two forms of data assimilation 8

2.2 The components of the EOS-MLS instrument. 12

2.3 The measurement suite for the EOS-MLS 14

2.4 Channel widths for EOS-MLS bands 16

2.5 The EOS-MLS radiometer layout 18

2.6 Observation paths for the EOS-MLS 20

2.7 A simple biological neuron 24

2.8 A sample neural network 25

2.9 The McCulluch-Pitts neuron 26

2.10 Input space for a 2-input AND problem 28

2.11 Input space for a 2-input XOR problem 29

2.12 A Simple Perceptron 30

2.13 A sample neural network with a hidden layer 32

2.14 Input space for a 2-input XOR problem a hidden node 33

2.15 A sample neural network with a bias node 36

2.16 The sigmoid function with bias 36

3.1 The network architecture 45

3.2 The temperature training set 49

3.3 The radiance training set 51

3.4 A sample network training result 53
3.5 A successful training run 55
3.6 Example error vs. hidden node number graph 57
3.7 The real error vs. hidden node number graph 59
3.8 A training run with weight decay 61
3.9 A training run with a nonlinear error function 62

4.1 Limb sounding geometry 66
4.2 A sample tangent pressure level profile 67
4.3 Tangent pressure variation 71
4.4 A neural network run using geometric heights 72
4.5 A neural network retrieval of tangent pressures 76
4.6 A neural network retrieval of tangent pressures using the reduced profile 77
4.7 Retrieved tangent pressures with no noise 80
4.8 Retrieved tangent pressures with $\sigma = 0.4$ K noise 80
4.9 Retrieved tangent pressures with $\sigma = 1.0$ K noise 81
4.10 Retrieved tangent pressures with $\sigma = 5.0$ K noise 81
4.11 Retrieved tangent pressures with $\sigma = 0.4$ K noise 83
4.12 Retrieved tangent pressures with $\sigma = 1.0$ K noise 83
4.13 Retrieved tangent pressures with $\sigma = 5.0$ K noise 84
4.14 A sample neural network run using varying tangent pressures . . . 85
4.15 A comparison between the neural network and Pumphrey's forward models with clean tangent pressures 86
4.16 A comparison between the neural network and Pumphrey's forward models with noisy tangent pressures 87

5.1 A neural network output for channel 16 of Band 1 93
5.2 A neural network output for channel 17 of band 1 94
5.3 A neural network output for channel 18 of band 1 95
5.4 A neural network output for channel 16 of band 1 trained with new data 97
5.5 A neural network output for channel 17 of band 1 trained with new data 98
5.6 A neural network output for channel 18 of band 1 trained with new data 99
5.7 A neural network output of channel 13 of band 1 100
5.8 A neural network output of channel 13 of band 1 trained with new data 101
5.9 The lower sideband of band 1 with oxygen, nitric acid, ozone and water vapour spectral lines 103
5.10 Sample radiance profiles from several channels in band 1 with no species 104
5.11 Sample radiance profiles from several channels in band 1 with water vapour and nitric acid 105
5.12 Sample radiance profiles from several channels in band 1 with water vapour, nitric acid and ozone 106
5.13 Neural network output for channel 3 of band 1 using ozone information 108
5.14 Example radiance profiles from band 7 109
5.15 The lower sideband of band 7 with oxygen, nitric acid, ozone and water vapour spectral lines 110

5.16 The upper sideband of band 7 with oxygen, nitric acid, ozone and water vapour spectral lines 110
5.17 A training run for Channel 1 of Band 7 112
5.18 A training run for Channel 13 of Band 7 113
5.19 A Typical Water Vapour Profile 114

6.1 The true Jacobian for channel 1 of band 1 123
6.2 The temperature section of the Jacobian from the neural network 124
6.3 The temperature section of the Jacobian from the neural network using hyperbolic tangent activation functions 126
6.4 The difference between the neural network and true Jacobians . . 127
6.5 The M values for the network generated Jacobian 128
6.6 The Jacobian for a single minor frame 130
6.7 The true Jacobian for channel 8 of band 1 131
6.8 The network generated Jacobian for channel 8 of band 1 132
6.9 The differences between the neural network and true Jacobians for channel 8 of band 1 133
6.10 The M values for the network generated Jacobian in Channel 8 . . 134
6.11 The true influence function for z 135
6.12 The influence function for z from the neural network 137
6.13 The influence function summed for the neural network 138
6.14 The real Jacobian for ozone in R3.B7F.C13 140
6.15 The neural network Jacobian for ozone in R3.B7F.C13 141
6.16 Error on the network generated Jacobian for ozone in R3.B7F.C13 142
6.17 The true Jacobian for H2O in R3.B7F.C13 143
6.18 The neural network Jacobian for H2O in R3.B7F.C13 145

A.1 An example no-hidden-layer perceptron 154
A.2 An example hidden-layer perceptron 157
A.3 Training a hidden-layer perceptron 159

List of Tables

2.1 A list of EOS-MLS radiometers with their corresponding bands and primary measurements 17

2.2 Differences between BNNs (Biological neural networks) and digital computers. 24

4.1 The scan points used from different channels to construct the reduced profile. 75

4.2 Training runs with noisy radiances in a cleanly trained network . 79

4.3 Training runs with noisy radiances in a network trained with noise 82

5.1 Instrument noise levels for channels of band 1 91

5.2 A list of channels that had difficulty previously with their new validation errors 96

7.1 A comparison of running times between the neural network and the true forward model 149

B.1 Definitions of quantities used 163

Acknowledgements

A number of people have helped in making the work carried out here possible. These people should be thanked, so here goes.

- **NERC** for providing the funding for the research to be carried out.
- **Prof. R. S. Harwood** for supervising the work, answering numerous questions and making me look at things in different ways.
- **Prof. H. C. Pumphrey** for the reference forward model used in training the neural network, helping fix numerous problems encountered with it as well as giving insight into its workings and answering numerous other questions.
- **Dr. L. Feng** for various discussions about how things work and comparisons with his own forward models.
- **Dr. C. Jiménez** for much background on neural networks and providing access to his work on neural networks within atmospheric science.
- **Many other people** within the department for all the help and support they provided throughout.

To these people, and to everyone else that I've forgotten, thanks.

I hereby certify that: (a) That the thesis has been composed by the candidate, and (b) That the work is the candidates own, and (c) that the work has not been submitted for any other degree or professional qualification

Chapter 1

Introduction

The aim of this thesis is to investigate whether a neural network can be used as a replacement for a traditional forward model for the EOS-MLS in a 4-D variational assimilation scheme. Data assimilation is the process of incorporating real-world measurements into atmospheric models, which provides optimal initial conditions essential for a successful forecast.

Numerical weather forecasting began in the 1940s when modern computers first became available (see e.g. Eliassen (1956)). Due to the speed of the computers of the time, forecasts were created for limited areas with a small number of grid points. Typically, they generated forecasts for 24 hour periods, due to the amount of time needed to run the simulations, and were based on observations made from ground stations scattered across the forecast region.

As these early forecasts were made only for short times at levels in the mid-troposphere, the models only calculated effects in the troposphere and used only temperature, pressure and the amount of water vapour as their forecast quantities. As computing power increased, the resolution of models was improved and the time-frame of forecasts could be extended.

In order to produce a successful forecast, two things are required. First, the laws that governing how subsequent states develop out of proceeding ones must be known. The second requirement is that the initial state of the atmosphere must be characterised as accurately as possible (e.g. Daley (1991)). Early forecasts used synoptic measurements produced from observation stations and radiosondes that were interpolated by hand to grid points for their initial state. As forecasting became more advanced, techniques were developed that allowed computers

to do this interpolation, not only in space but also in time, allowing asynoptic measurements to be incorporated. The process of adding the measurements to produce this initial state was named data assimilation. Since it was first introduced, the term data assimilation has grown to include many methods of adding real-world measurements into dynamic models (e.g. Lorenc (1986)).

From the 1960s onward, weather satellites have been launched to help aid understanding of the atmosphere. Instruments on satellites can be used to calculate the temperature and moisture profiles throughout the atmosphere, which can be used within numerical weather models to help improve them further. Satellite data sets have some advantages over ground-based and radiosonde data as they provide much better horizontal coverage and resolution, filling in gaps between ground stations that are often hundreds of kilometres apart.

As computing power increased to the stage where longer forecasts could be provided, the state of the upper atmosphere began to play an important role in numerical forecasting. Models of the stratosphere were developed and added to the forecast models. Several of the more influential chemical species, such as ozone, were also added to the model. Today, information from nadir sounding satellites is routinely assimilated into forecast models in the form of profiles. While this gives much improved accuracy in forecasts, there are some problems with this approach.

Satellite retrievals generally work by using an optimal estimation method (e.g. Rodgers (2000)). An a-priori profile for the desired products is supplied (from climatology data), in order to provide a starting point for the retrieval, and the retrieval system calculates the expected radiances from these and the Jacobian for the forward model (the derivatives of the radiance vector with respect to the state vector) at that point. The difference between the true radiances and the generated radiances is then used with the Jacobian to update the state vector and the radiances are regenerated. This process is repeated until the maximum number of iterations has been achieved(e.g. Livesey et al. (2006)). While this produces good results, there will always be an element of the a-priori profile left in the system, which is undesirable. The retrieval process also introduces delays in getting the data into the assimilation scheme.

Assimilating radiances directly into a numerical model solves the problem of a-priori information. In this case, the retrieval is effectively performed by

the assimilation with the model fields acting as the a-priori. This means the resulting error will be between the original model state and the true atmosphere state, instead of introducing the effects from a state that might be totally unlike either (e.g. Lorenc (1986)). Using radiances directly in an assimilation is known as *direct assimilation.*

With the introduction of the stratosphere and more vertical levels into forecast models, a need arises for instruments that can provide data at a range of heights throughout the atmosphere. Nadir sounding instruments are limited in their vertical resolution and can only provide data at a limited number of heights. By contrast, limb sounders look at a tangent to the planet's surface and can provide data at a wide range of heights.

One instrument that is of use here is a microwave limb sounder (Janssen (1993)). The first satellite-borne microwave limb sounder (UARS-MLS - Barath et al. (1993)) was launched in September 1991 and the second generation (EOS-MLS - Waters et al. (2006)) was launched in July 2004. Along with temperature and pressure profiles through-out the troposphere and stratosphere, it also provides profiles for a number of chemical species.

As the assimilation process uses a lot of computing power, there is only a small amount of time available to carry out the forward model each time-step. As satellite instruments grow in complexity, the forward models for them (used to generate expected radiances) also grow in complexity, requiring more computing time.

To counter this, forward models are often linearised when used in assimilation, sacrificing precision for speed. This method relies on changes in radiances being nearly linear for small changes in the forward model inputs. However, this does not work in some cases when the radiance response is not linear enough and a new method of speeding up the forward model must be found.

One possible solution to this non-linearity problem is to use neural networks (e.g. Jain et al. (1996)). A neural network can be considered as a non-linear fitting technique with the inputs and outputs of the algorithm represented as a pair of vectors. The algorithm also uses an intermediate vector at a so-called "hidden layer". Each element of this intermediate vector is associated with a "node" at which other ancillary informations, "weights", are used in the calculation.

To find the weights that a neural network needs, it is necessary to train it on

a set of input-output vector pairs, found by other means. During this training process, the network calculates the error on the outputs for the training set and updates its weights according to the its training rules. Once the error on the outputs is low enough, training is stopped and the weights are fixed. After this, running the network is cheap in terms of computing power as it only consists of a series of additions and multiplications (e.g. Sarle et al. (1997)).

A neural network does have some limitations that do not exist in a traditional forward model. A neural network is very good at interpolation but bad at extrapolation. To ensure the network works well the training set must include profiles from across the whole range of expected inputs / outputs. Checks must also be made when running that the profile is indeed within expected ranges. If these checks are not done, a neural network may produce wrong results. As neural networks are trained prior to use, any parameters not included as inputs must be fixed. For example, a neural network trained on frequency range of an instrument cannot be used to generate radiances for a different frequency range without retraining, due to different instrument responses and different chemical species affecting radiances.

In this thesis, a neural network forward model is constructed for the EOS-MLS for use in an assimilation scheme as an investigation of the feasibility of using such neural networks in direct assimilation. As the aim is to use this forward model in an assimilation scheme, the majority of the thesis will deal with radiances that are only affected by temperature changes. The thesis is split into seven chapters. The second chapter presents background information about the EOS-MLS, data assimilation and neural networks. It also gives details of previous work carried out involving neural networks in atmospheric science.

Chapter three gives details of what is required of a neural network in this case. It shows that a neural network can simulate a forward model in an idealised situation in which only temperature affects the radiances. Chapter four introduces the problem of tangent pressures, which are related to determining the pointing information associated with each observation. It demonstrates what tangent pressures are and why they are a problem in an assimilation scheme. It then gives a method for dealing with tangent pressures outside an assimilation scheme and shows how they can be incorporated into the neural network forward model.

In chapter five, the neural network forward model is extended to more than one channel and additional chemical species are added into the forward model. Chapter six explores how a Jacobian of a neural network can be found and how it compares to the true forward model's Jacobian. Finally, chapter seven discusses the main conclusions of the thesis and looks at ways it can be extended. Appendix A gives a list of acronyms and symbols used in this thesis.

Chapter 2

Background

2.1 Introduction

This chapter introduces to the three main technologies used within this thesis: Data assimilation, the EOS-MLS and neural networks.

The data assimilation section gives details of the particular type of data assimilation that the system is designed to be used in. The EOS-MLS section gives details of what the instrument is and how it operates and the section on neural networks describes the type of neural network used within this project, as well as giving a general overview of other types.

The final section also gives details of previous work using neural networks within atmospheric science.

2.2 Data Assimilation

Data assimilation is the method of taking real world measurements and incorporating them into a model. This technique is used extensively in the atmospheric science community. This is done as forecast models need the most accurate initial state of the atmosphere possible in order to accurately predict future states. There are many varieties of data assimilation. Here, the fundamental concepts of data assimilation will be introduced and then one type of data assimilation, 4D-Var, will be discussed. Further information about other types is available in e.g. F. Bouttier (1999). This section is derived from the paper by Nichols (2002), which gives an introduction to different forms of data assimilation.

2.2.1 Fundamental Concepts

In atmospheric modelling, the models are usually very large - of the order of 10^7 state variables and growing as computers get more powerful. As the atmosphere is a chaotic entity, there is no way to accurately set up a model to reflect the true state of the atmosphere at any particular point. Instead, an approximation is derived from observations and is used as the initial state for the model. Typically, these observations come from a variety of sources - radiosondes, weather stations, radar and satellites, giving of the order of $10^5 - 10^6$ observations per day. Once the initial state has been set, the model is run for a prescribed time. After this, the model's state is reinitialised using new observations, combined with the current model state, and the model is run again.

There are two classes of assimilation scheme which can be applied - sequential assimilation and four dimensional assimilation, which are illustrated in figure 2.1. In sequential assimilation, the model is started with an *a-priori* estimate for the initial state, and is evolved to a later time, t_k, where the first observation is made (e.g. Daley (1991)). The model state at this time is known as the background field. This background state is used to create a predicted observation vector which can be compared to the true observation vector. The difference between the predicted and true observation vectors is then used to update the background state to get an improved model state, called the analysis field. This can be done in a variety of different ways, such as "nudging" the background state towards the observations or a variational process where the initial state is altered to bring the background state nearer the observations (3D-VAR). From this point, the model is evolved again until the next time when observations are available.

Sequential assimilation incorporates many popular forms of assimilation. As more computing power becomes available and the number of available observations increase, another form of assimilation is becoming more popular - four dimensional assimilation.

Four dimensional assimilation considers all the observations available within a time window to give improved estimates over that window. This allows a range of observations in time to have influences on the analysis, which results in more consistent forecasts. The major form of four-dimensional assimilation is 4D-VAR (e.g. Daley (1991)), which is discussed here.

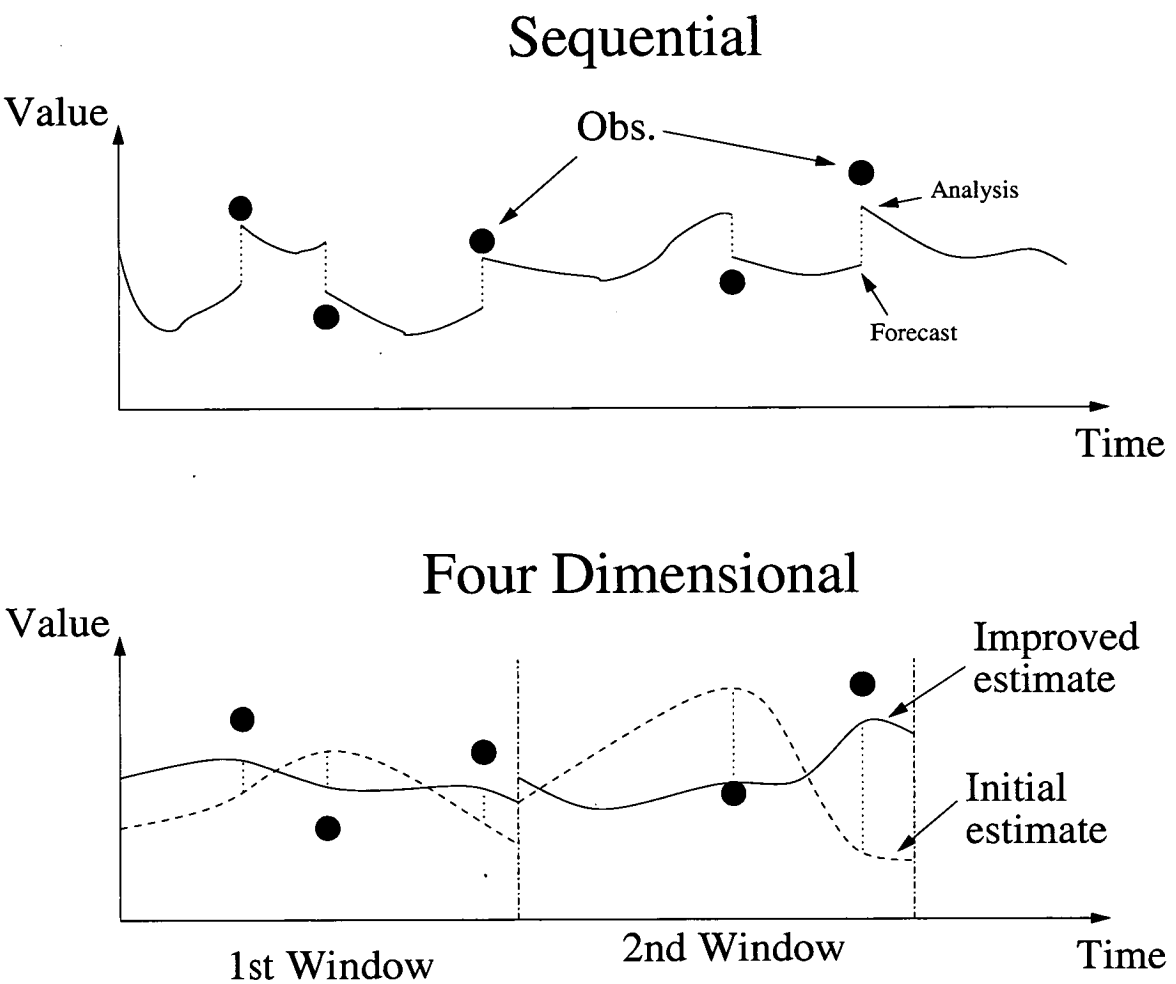

Figure 2.1: Two forms of data assimilation - sequential and four-dimensional assimilation. In the sequential case, the model is evolved to the time an observation is made. A correction is made to the model state to account for this observation and the model evolution is continued. In four-dimensional assimilation, all the measurements in a prescribed time window are used to provide improved estimates for all the states in that window.

2.2.2 4-Dimensional Assimilation

Here, a brief treatment of 4D-VAR is presented. Further discussion about 4D-VAR can be found in e.g. Nichols (2002). In an assimilation system, the model is described by discrete non-linear equation 2.1 where $\vec{x}_k$ is the model states, $\vec{u}_k$ are the known forcing inputs and f_k is the (non-linear) function describing the evolution of the system. The observations are related to the system states by way of equation 2.2, where the forward model, h_k, is a non-linear function and the error term, δ_k, is assumed to be unbiased, uncorrelated in time and Gaussian with covariance matrix R_k.

$$\vec{x}_{k+1} = f_k(\vec{x}_k, \vec{u}_k), \qquad k = 0, \ldots, N-1 \tag{2.1}$$

$$\vec{y}_k = h_k(\vec{x}_k) + \vec{\delta}_k \tag{2.2}$$

Background estimates for the initial state, $\vec{x}_0^b$ are assumed to be known with the initial random error assumed to be Gaussian with covariance matrix B_0. The observation errors and the background errors must be uncorrelated. Using these facts, the data assimilation problem can be restated as *"Minimise, with respect to* $\vec{x}_0$ *the cost function (equation* 2.3*) subject to* $\vec{x}_k$, $k = 1, \ldots, N-1$, *satisfying the system equation* 2.1 *with initial states* $\vec{x}_0$*"*. Minimising the cost function, equation 2.3, involves simultaneously trying to get the new starting state, $\vec{x}_0$, to be near the background starting state, $\vec{x}_0^b$, while trying to get the predicted observations as close to the true observations as possible.

$$J = \frac{1}{2}(\vec{x}_0 - \vec{x}_0^b)^T B_0^{-1}(\vec{x}_0 - \vec{x}_0^b) + \frac{1}{2}\sum_{k=0}^{N-1}(h_k(\vec{x}_k) - \vec{y}_k)^T R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k) \tag{2.3}$$

Two assumptions are then made. The first assumption is that the states of the model, $\vec{x}_k$, can be expressed in terms of the initial state, $\vec{x}_0$, as $\vec{x}_k = f_k(f_{k-1}(\ldots f_0(\vec{x}_0, \vec{u}_0)))$. The second assumption is that both f_k and h_k can be linearised around the current trajectory, using equations 2.4 and 2.5, where F_k and H_k are the Jacobians of f_k and h_k with respect to $\vec{x}_k$.

$$\vec{x}_{k+1} = f_k(\vec{x}_k, \vec{u}_k) + F_k\vec{\epsilon}_k \tag{2.4}$$

$$h_k(\vec{x}_k) - \vec{y}_k \approx F_k H_k \vec{\epsilon}_{k-1} - \vec{y}_k \tag{2.5}$$

Using these relations, along with the constraints given by equation 2.1, the gradient of the cost function can be derived as in equations 2.6 - 2.9, where $d_k = R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k)$ is called the departure of the observation and $\nabla_{\vec{x}_0}$ is the derivative with respect to $\vec{x}_0$.

$$\nabla_{\vec{x}_0} J = \quad \nabla_{\vec{x}_0} J_0 + \sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i \tag{2.6}$$

$$\nabla_{\vec{x}_0} J = \quad B_0^{-1}(\vec{x}_0 - \vec{x}_0^b) + \sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i \tag{2.7}$$

$$\sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i = \quad \sum_{k=0}^{N-1} F_1^T F_2^T \ldots F_k^T H_k^T d_k \tag{2.8}$$

$$\sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i = H_0^T d_0 \quad + \quad F_1^T (H_1^T d_1 + F_2^T (H_2^T d_2 + \ldots F_{N-1}^T H_{N-1}^T d_{N-1})) \tag{2.9}$$

Defining λ_k as equation 2.11, the gradient of the cost function can be rewritten as equation 2.12. λ_k are the adjoint variables, which measure the sensitivity of the gradient to changes in the measurement k.

$$\lambda_N = 0 \tag{2.10}$$

$$\lambda_k = F_k^T(\vec{x}_k)\lambda_{k+1} \quad -H_k^T R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k) \tag{2.11}$$

$$\nabla_{\vec{x}_0} J = B_0^{-1}\left(\vec{x}_0 - \vec{x}_0^b\right) - \lambda_0 \tag{2.12}$$

Each iteration, one forward solution of the model equations (2.1 - 2.2) and one backward solution of the so-called adjoint equations (2.10 - 2.11) is computed using the best current estimate of the initial state. The initial state is then updated using a gradient descent approach.

2.2.3 What is Needed for a 4D-VAR Assimilation Scheme

In order to incorporate measurements from an instrument into a 4D variational assimilation scheme, several things are required in practise. The first, and most important, is a fast forward model.

Running a 4D variational assimilation scheme is expensive in terms of computer time. For each assimilation window, a forward run of the assimilation model, as well as a backward run of the adjoint equations is required for each iteration, and there may be several iterations. Within this, the forward model of the instrument must also be run each iteration for each measurement in order to simulate the instruments response to the new state of the atmosphere. Since there may be many measurements to be assimilated, the forward model for each instrument is only given a small amount of time to run.

The second thing needed is the Jacobian of the forward model (H_k in equation 2.11), which is used to update the model state vector. This must be calculated every time the forward model is run and can be generated either by differentiating the forward model by hand, or by automatic differentiation techniques available in a number of computer programs (e.g. Giering and Kaminski (1998)).

The final thing that needs to be supplied in order to assimilate measurements from an instrument is an estimate of the error characteristics. This is in the form of the error covariance matrix for the instrument and includes instrument errors, errors introduced due to inaccuracies in the forward model and interpolation errors.

2.3 The EOS-MLS

The EOS-MLS is a microwave limb sounding instrument (Waters et al. (2006)) aboard the EOS Aura satellite (Schoeberl et al. (2006)) which was launched on 15th July 2004. It is the successor to another instrument called the UARS-MLS which flew on the UARS satellite during the 1990's. The instrument's main aim is to observe atmospheric chemistry in the stratosphere and upper troposphere.

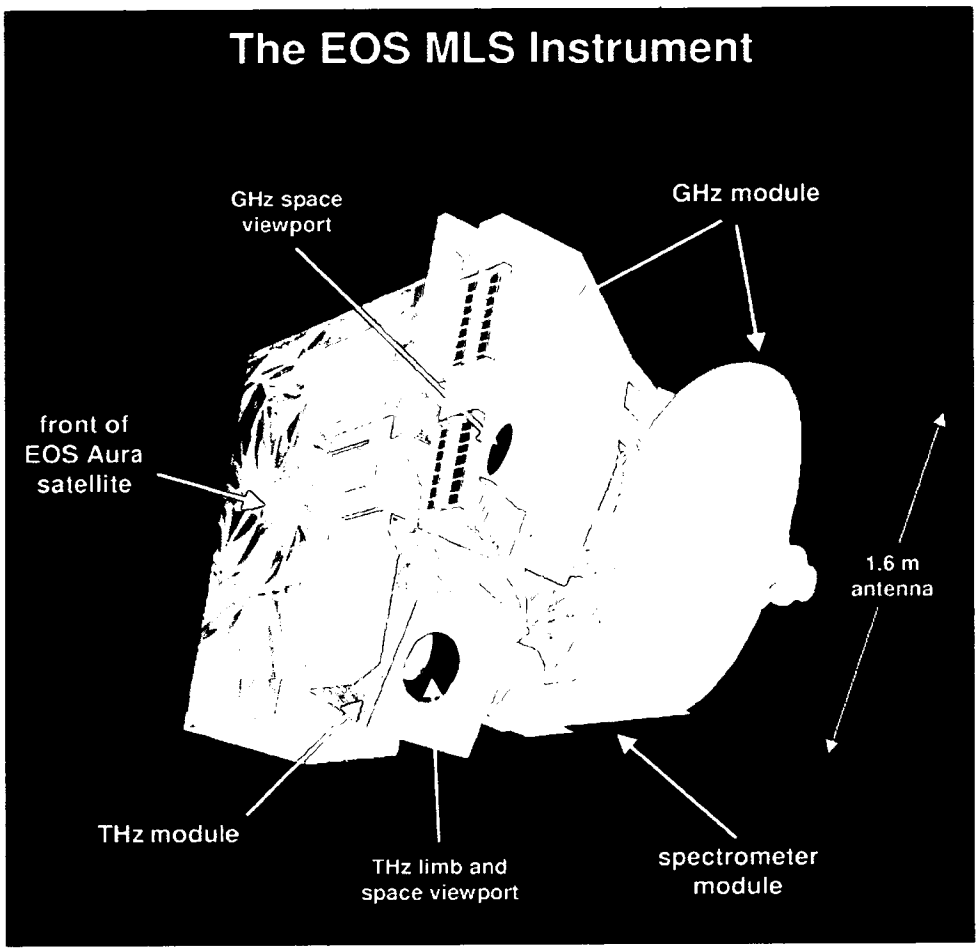

Figure 2.2: The components of the EOS-MLS instrument.

2.3.1 Instrument Details

The EOS-MLS is a passive microwave limb sounding instrument that points along the orbital motion. A diagram showing the basics of the instrument is given in figure 2.2. While travelling, the field of view of the instrument is scanned upward from 2.5km to 62.5km[1], creating a series of 120 radiance measurements per channel in one scan. Each measurement within a scan is known as a *minor frame* and one complete set of measurements is called a *profile*. A profile plus calibration information is called a *major frame*. On the ray at the centre of the field of view, the pressure at the point closest to the Earth is called the *tangent pressure*.

[1]For the GHz radiometers. The THz radiometer is different and is described in Pickett (2006)

The Aura satellite is in a 98° orbit at a height of 705km. It is a sun synchronous satellite with an orbital period of approximately 100 minutes. Each scan takes 24.7s, resulting in 240 scans per orbit and around 3500 scans per day. The instrument resolution at the limb tangent point is typically around 3 km vertically, 5 km cross-track and 500 km along-track. This gives the instrument excellent vertical resolution at the cost of horizontal resolution when compared to nadir sounding instruments.

The instrument is the successor to the UARS-MLS which flew on the UARS satellite during the 1990's. The EOS-MLS improves on the UARS-MLS in a number of ways, primarily, it covers more chemical species in more bands with better resolution. For a comparison between the EOS-MLS and UARS-MLS instruments, see Waters (1999).

The instrument can measure a number of chemical species including ozone and water vapour as well as several other quantities such as the temperature. An indication of the measurement suite can be found in figure 2.3.

The instrument has a set of 34 bands split over 5 radiometers, measuring a range of frequencies from 118 GHz to 2.5 THz (0.1 - 3 mm wavelength). Each band is centered on a spectral emission line and consists of a number of channels. There are 4 different types of bands - full-width, mid-width, narrow and wide. Full-width bands consist of 25 channels and cover a region of 1300 MHz and allow useful measurements in the atmospheric pressure range from $\sim$ 100 hPa to $\sim$ 1 hPa. Mid-width bands are 11 channels wide and cover 200 MHz, providing additional measurements in the upper stratosphere ($\sim$ 10 hPa to $\sim$ 1 hPa). Narrow bands have 10 MHz resolution and cover narrow spectral lines at atmospheric pressures less than $\sim$ 1 hPa. They have 129 channels and are implemented as Digital Autocorrelator Spectrometers (DACS). Wide bands are bands of 4 channels that extend full-width bands down into the troposphere. Each channel in a wide band is 0.5 GHz wide.

The channel width for different band types varies between 500MHz (wide band channels) and 0.15MHz (narrow band channels). The channel width in full- and mid-width bands varies with channel number and channel width for these band types is shown in figure 2.4. This figure shows typical radiance values for a simulated oxygen line at a height of $\sim$ 10hPa. The width and position (relative to the band center) of individual channels are shown by the horizontal lines. Mid-

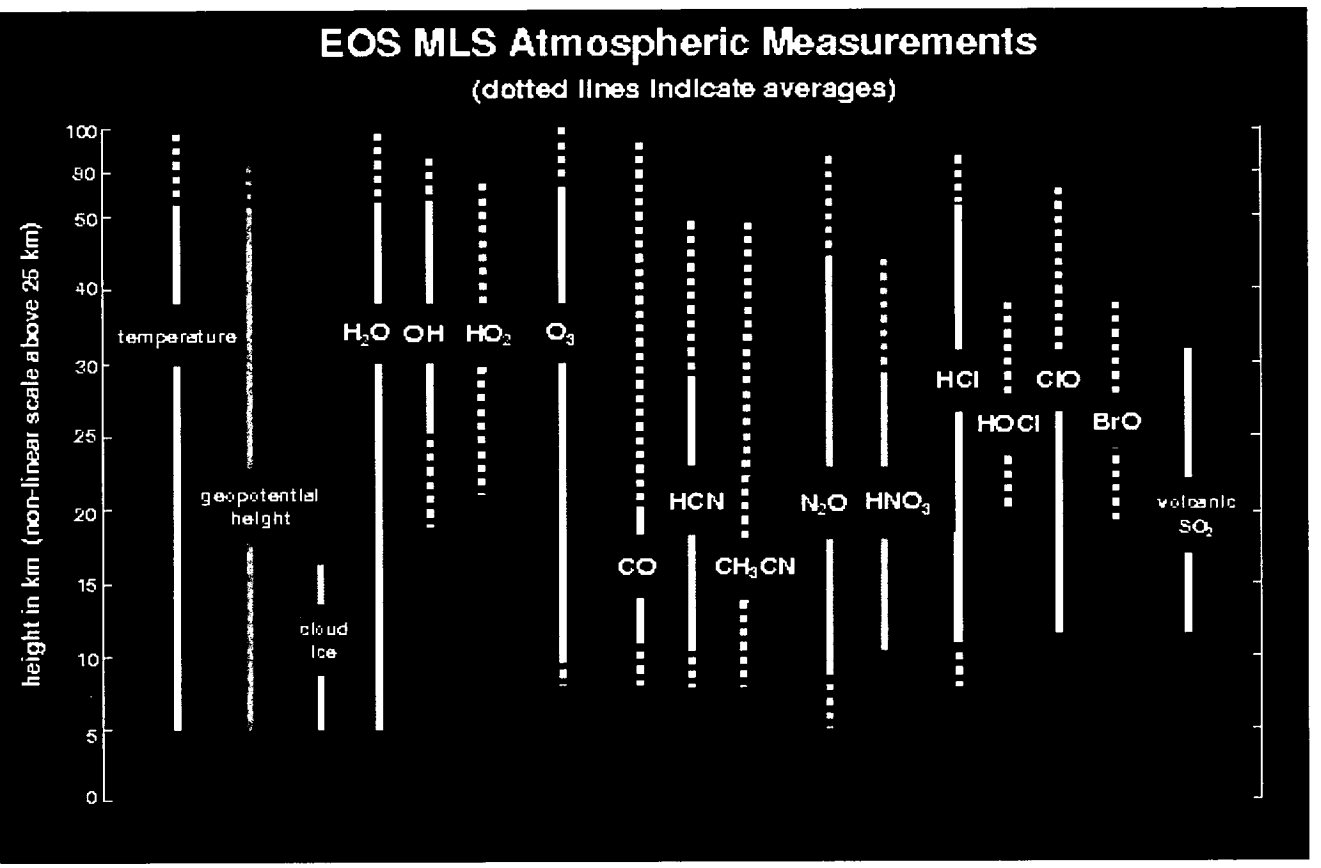

Figure 2.3: The measurement suite for the EOS-MLS. This shows an indication of the species the EOS-MLS instrument can measure and typical heights these species can be measured at. Solid lines indicate useful precision for individual profiles. Dotted lines indicate that zonal averages are needed for useful precision.

width bands consist of channels between the dotted lines in the figure. As the individual channels are much narrower than spectral lines, they can be treated as monochromatic in calculations.

A full list of all bands giving details of radiometer, type of band and main target is given in table 2.1. Channels are reference as (Radiometer).(Band).(Channel). So, R1A.B1F.C1 means "channel 1 of band 1 of radiometer 1A". The "F" in the band indicates it is a full-width band. Other band types are denoted by "W", "D" and "M" for wide, narrow (DACS) and mid bands respectively. More detailed information about the EOS-MLS nomenclature can be found in Livesey and Wu (1999). This information is represented graphically in figure 2.5 which shows the measuring frequencies of all radiometers for the EOS MLS, centered around the local oscillator frequencies for the radiometer. As radiometers 2 - 5 use split sideband, the locations of both sidebands are presented on the graph. Radiometer 1 uses a single sideband, with the upper sideband filtered out.

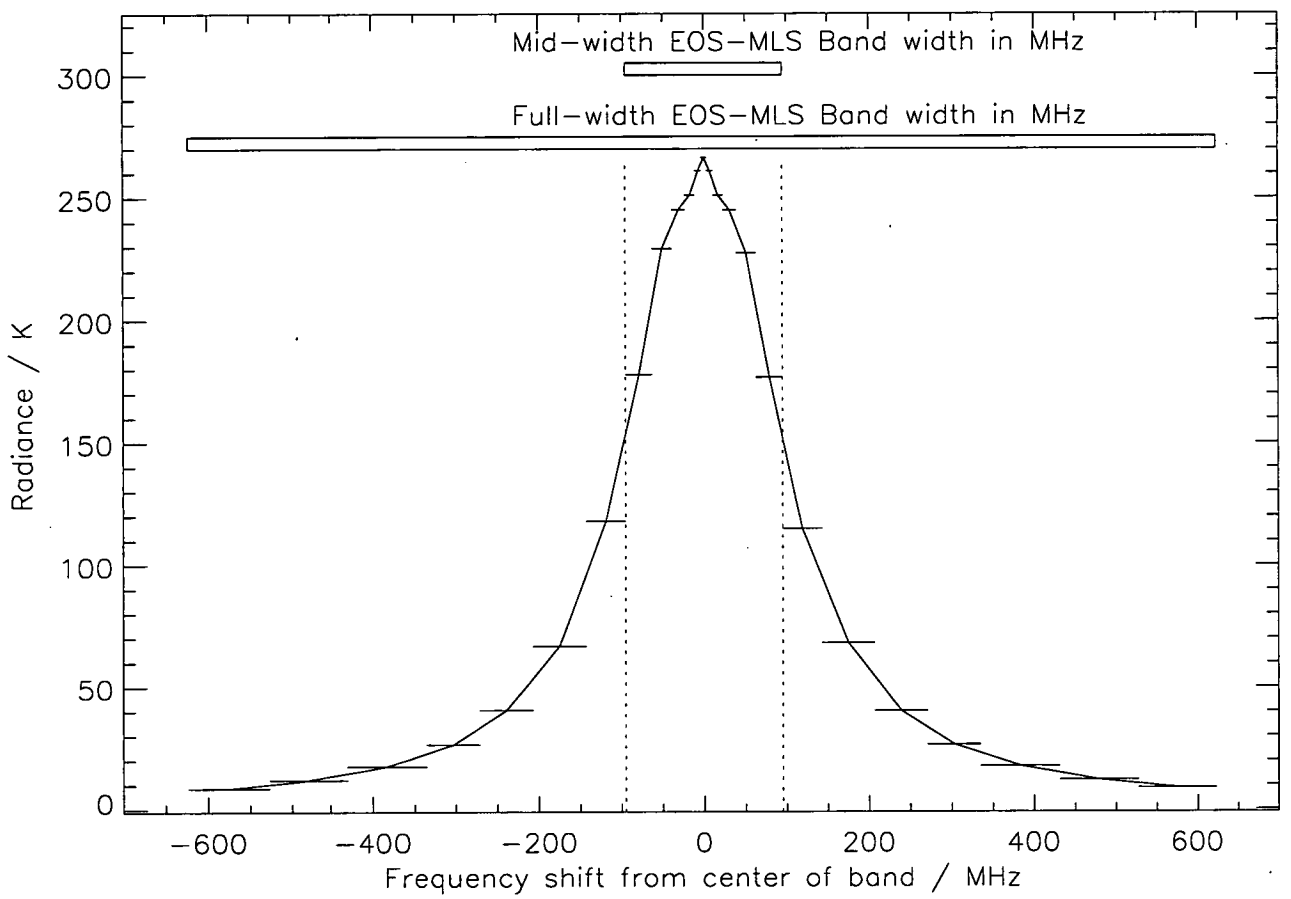

Figure 2.4: Channel widths for full- and mid-width bands of the EOS-MLS. The vertical scale gives example radiance values for this band at 100hPa. The horizontal lines show individual channel widths, which range from 96MHz at the edges to 6MHz for the central channel. Mid-width band channels are those between the dotted lines.

Radiometer 1st LO frequency	Band	Type	Primary Measurements	Spectral Line Frequency(GHz)
R1 126.8000 GHz	B1	F	P / T	118.75
	B21	F	P / T	118.75
	B22	N	P / T	118.75
	B26	N	P / T	118.75
	B32	W	P / T	118.75
	B34	W	P / T	118.75
R2 191.9000 GHz	B2	F	H_2O	183.31
	B3	F	N_2O	200.98
	B4	F	HNO_3	181.59
	B5	F	ClO	204.35
	B6	F	O_3	206.13
	B23	N	H_2O	183.31
	B27	M	HCN	177.26
R3 239.6600 GHz	B7	F	O_3	235.71
	B8	F	P / T	233.94
	B9	F	CO	230.54
	B24	N	O_3	235.71
	B25	N	CO	230.54
	B33	W	O_3	235.71
R4 642.8700 GHz	B10	F	ClO	649.45
	B11	F	BrO	650.18
	B12	F	N_2O	652.83
	B13	F	HCl	625.92
	B14	F	O_3	625.37
	B28	M	HO_2	649.70
	B29	M	$HOCl$	635.87
	B30	M	HO_2	660.49
	B31	M	BrO	624.77
R5 2522.7816 GHz	B15	F	OH	2514.32
	B16	F	OH	2509.95
	B17	F	P	2502.32
	B18	F	OH	2514.32
	B19	F	OH	2509.32
	B20	F	P	2502.32

Table 2.1: A list of EOS-MLS radiometers with their corresponding bands and primary measurements. Types are shown by one of 4 symbols: F represents full-width bands, W are wide bands (Individual filters), M are mid-bands and N are narrow-bands (DACS). P/T indicates the band is centered on an Oxygen line and its primary measurements are pressure / temperature.

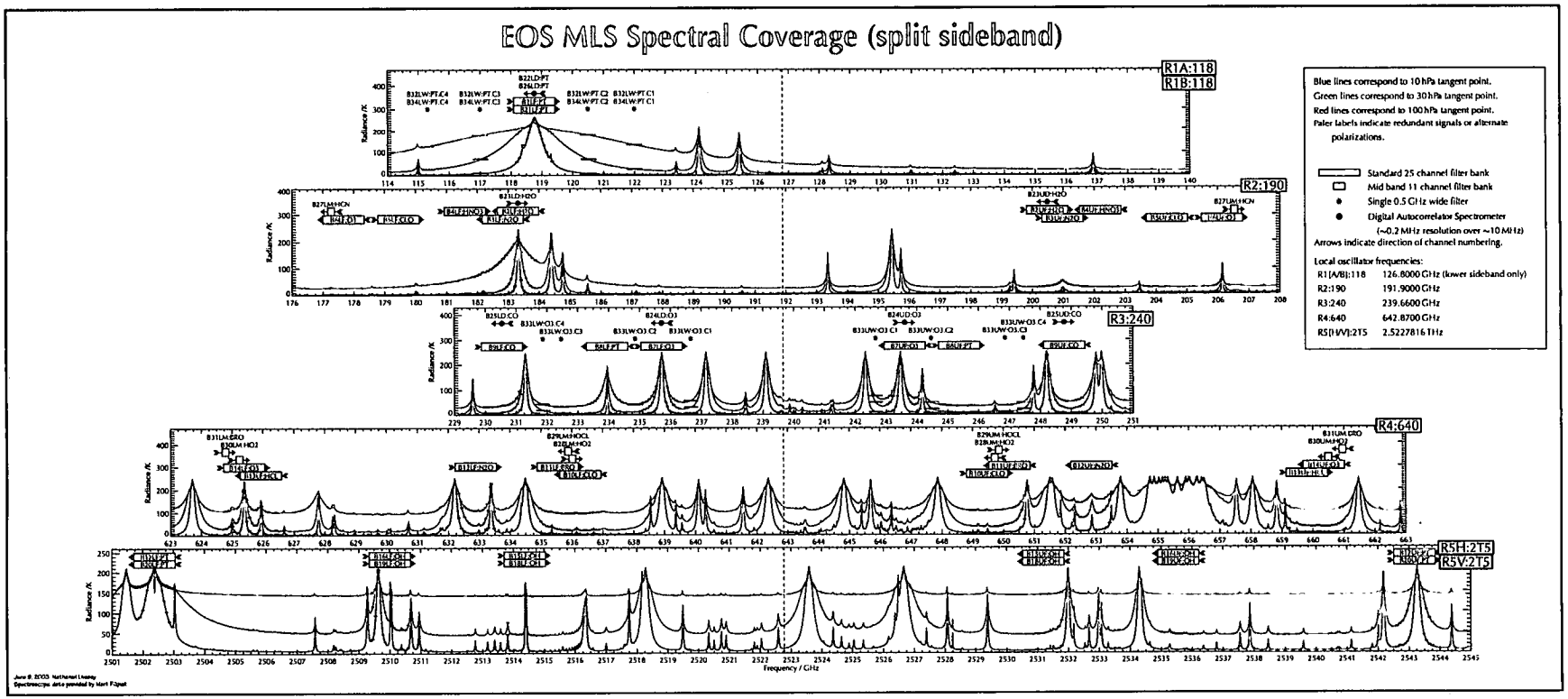

Figure 2.5: The bands measured by the EOS MLS and the spectral coverage of all radiometers, centered on the local oscillator frequencies. The three spectra in each panel correspond to nominal atmospheric radiances for tangent pressures of 10, 30 and 100 hPa and assume single sideband response for R1 and equal relative sideband responses on all other radiometers.

18

2.3.2 Measurements

Data from the EOS-MLS instrument undergo several levels of processing. The data at each stage are labelled levels 0 to 3. Each of these is explained in brief below.

The first level, level 0, is the raw telemetry data sent back by the instrument. This includes the raw counts and information about where the instrument is looking (the FOV).

Level 1 data are data that have had some processing work done to convert the raw telemetry into more useful information, such as the latitude and longitude of the measurement (from the satellite data) and the geometric tangent height of each minor frame. The calibrated radiances are also generated at this stage, taking into account various external factors such as antenna emissions and scattering effects. These calibrated radiances are checked to determine if any are unusable (i.e. are obviously wrong or out of expected measurement range). If any unusable radiances are found, a flag is set stating this.

Level 2 files contain the retrieved profiles. There are generally in the form of a set of values for the species involved, on a fixed pressure grid, for each profile. Level 2 data also include the tangent pressure levels of the radiances (discussed in section 2.3.1).

Level 3 files are made up of monthly means of zonal means for different species and other mapped products. These are not relevant to the current study.

2.4 The Radiative Transfer Equation

The general solution of the radiative transfer equation in the case of microwave radiometry can be written as equation 2.13, provided the atmosphere is in local thermal equilibrium and no cloud particles are present (see e.g. Janssen (1993)). In this case, scattering is neglected as the wavelengths involved are typically much larger than the diameter of aerosol molecules.

$$I(\nu, \Omega, \vec{x}) = I_\infty(\nu)e^{-\tau(\infty,\nu)} + \int_0^\infty k(\xi, \nu, \vec{x})B(\nu, T)e^{-\tau(\xi,\nu)}d\xi \qquad (2.13)$$

Here, I is the spectral radiance as a function of the solid angle, Ω. ν is the frequency, ξ the distance along the observation path, I_∞ the intensity at the

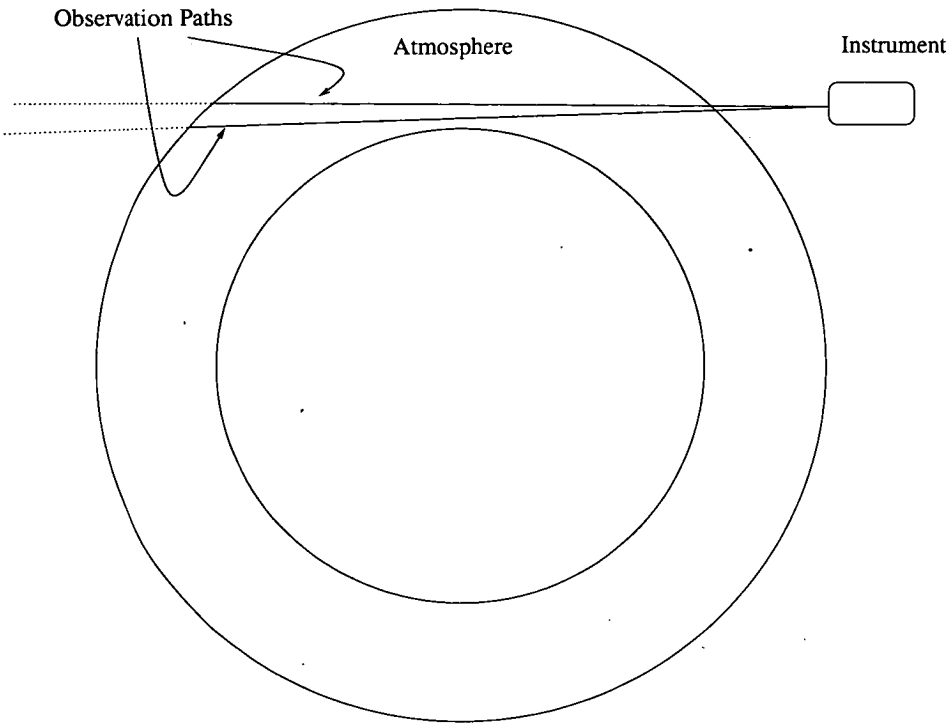

Figure 2.6: Observation Paths for the EOS-MLS. This shows two observation paths for a limb sounder. The radiance can be calculated by summing the contributions along the solid part of the observation line. The dotted part of the line can be considered to contribute nothing to the final radiance and so can be replaced with a constant value of the cosmic background radiation level.

end of the observation path, $k(\xi, \nu, \vec{x})$ the total absorption coefficient, between the height ξ and the top of atmosphere, summed over the all species in the state vector, $\vec{x}$. T the physical temperature, $\tau(\xi, \nu)$ the optical depth defined by equation 2.14 and $B(\nu, T)$ is the Planck function defined by equation 2.15 where h is the Planck constant, c the speed of light and k_B the Boltzmann constant.

$$\tau(\xi, \nu) = \int_0^{\xi} k(\xi, \nu) d\xi \tag{2.14}$$

$$B(\nu, T) = \frac{2h\nu^3}{c^2} \frac{1}{e^{h\nu/k_B T} - 1} \tag{2.15}$$

In the case of microwave limb sounding, the observation path can be considered to end at the edge of the atmosphere (figure 2.6), and the value of I_∞ will be the cosmic background radiation value.

In order to find the measured instrument radiance for a particular channel, equation 2.13 must then be combined with the instrument response function, $G(\Omega, \Omega_0(t), \nu)$, for that channel and the field of view (FOV) function, $\Phi(\nu)$ (Read et al. (2004)). This is then averaged over the frequency range of the channel. Due to the continuous nature of the scan, this must also be averaged over the solid angle over which the FOV function is measured. This results in two expressions, one for the upper sideband of the channel and one for the lower sideband, equation 2.16 and 2.17 respectively. $\Omega_0(t)$ is the FOV direction that varies over time, t. Ω_A is the portion of the solid angle over which the instrument response is measured.

$$I_{USB} = \frac{\int_{\nu_{lo}}^{\infty} \int_{\Omega_A} I(\nu, \Omega, \vec{x}) \Phi(\nu) G(\Omega, \Omega_0(t), \nu) d\Omega d\nu}{\int_{\nu_{lo}}^{\infty} \int_{\Omega_A} \Phi(\nu) G(\Omega, \Omega_0(t), \nu) d\Omega d\nu} \tag{2.16}$$

$$I_{LSB} = \frac{\int_{-\infty}^{\nu_{lo}} \int_{\Omega_A} I(\nu, \Omega, \vec{x}) \Phi(\nu) G(\Omega, \Omega_0(t), \nu) d\Omega d\nu}{\int_{-\infty}^{\nu_{lo}} \int_{\Omega_A} \Phi(\nu) G(\Omega, \Omega_0(t), \nu) d\Omega d\nu} \tag{2.17}$$

These can then be combined and averaged over the scan time using fractional ratios, r_u and r_l which take into account the loss of signal as a result of scattering, spill-over, absorption and efficiency of the receiver. Finally, equation 2.18 gives the level 1 radiances, denoted by $\dot{I}$. Here, I_{bsl} is an additional term that corrects the result for various additional effects outside the intended measurement. Further information about this and how r_u and r_l are defined can be found in Read et al. (2004).

$$\dot{I} - I_{bsl} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \{r_u I_{USB} + r_l I_{LSB}\} \, dt \tag{2.18}$$

The integrals in the denominators of equations 2.16 and 2.17 are normalisations of the instrument response functions and can be considered "constant" and folded into $\Phi(\nu)$ and $G(\Omega, \Omega_0(t), \nu)$. The integration over Ω_A is used to normalise antenna gain over Ω_A and evaluates to a constant. All the functions in equations 2.16 and 2.17 are channel dependent and it is assumed that the antenna response is frequency independent across the highly weighted part of the filter response, but different for the two sidebands.

As mentioned in section 2.3.1, radiometer 1 uses only the lower sideband, with the upper sideband filtered out. In this case, equation 2.18 becomes 2.19 where

r_l is still required to account for loss of signal as before.

$$\dot{I} - I_{bsl} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \{r_l I_{LSB}\}\, dt \tag{2.19}$$

2.5 Neural Networks

Research into artificial neural networks began in the 1940s, when they were hailed as the next big thing in computing. This interest swiftly died out as technical problems arose, but in the early 1980's these problems started to be solved and interest was rekindled. Today neural networks are used in many different applications and fields, from helping to fly helicopters (Buskey et al. (2001)) to analysing MRI images in hospitals (Feltham and Xing (1994)).

In theory, artificial neural networks, or ANNs, can do anything a conventional computer can do, plus more (Sarle et al. (1997)). They are well suited to pattern recognition and classification tasks, which conventional programs have difficulty with. They are also particularly suited to problems that are not exactly soluble by tradition methods but have many examples of input/output sets.

There are literally thousands of different types of ANNs in the world today, with a new variation being created all the time (e.g. Sarle et al. (1997)). Each network created is virtually unique, with its own learning rules, network structure and its own quirks which makes building a neural networks almost as much an art form as a science. Having said this, all ANNs can be split broadly into one of two groups.

The first group uses supervised training. In this case, the training set consists of a series of input-output pairs and the aim is to minimise the difference between the true outputs and the network outputs by altering values of some of the internal parameters of the network. This type of network is used extensively in "computational neural networks".

The second major group uses unsupervised learning. Here, no output is given during training and the aim is to classify inputs according to characteristics within the input values. An example of this is an insect classification system. In this system, a collection of insects would be measured in various ways i.e. the wing span, colour and overall length may be measured and encoded. These parameters form the training set. The task for the network would be to classify the insects

into different species, which is achieved by calculating the "distance" between the input vector and a set of idealised vectors for each classification group (calculated during the training cycle). This method is used in image recognition and so-called "Boltzmann Machines". Here, only computational neural networks are discussed.

This section assumes some knowledge of the fundamentals of neural networks. Appendix A contains a more detailed and thorough outline of how the principle types of network described here (simple and multi-layered perceptrons) operate.

2.5.1 Background

The ideas for artificial neural networks (ANNs) are inspired by their biological counterparts (the brain). It is therefore useful before looking at ANNs to understand the basics of how the brain works (e.g. Rojas (1996)).

The brain is made up of approximately 10^{14} neurons. Each of these neurons is connected to up to 10^4 other neurons by means of dendrites and synapses. Dendrites gather the inputs from other neurons while synapses send processed information out to other neurons. Between these are two components, first the soma which processes all the inputs from the dendrites, then the axon, which converts these into the output for the synapses. This is shown in figure 2.7.

In this way, the brain can be thought of a massive parallel computer, with 10^{14} parallel processors, that are only capable of processing a few simple commands, unlike conventional computers that only have a few processors (typically 1) but can perform many different tasks.

There are many differences between conventional computers and the brain. The main difference, apart from the number of processors, is the way memory is stored and addressed. In conventional computers, the memory is a physical block that is referenced by address. If a program asks for a certain memory address, the computer will return the contents of that address with no checks to ensure the data in it are sensible. In biological neural networks, the memory is stored within neurons and can be retrieved when a partial or corrupt version of the information is passed through the network (Braspenning et al. (1995)). This is known as "content addressable" and has the highly desirable property, for some applications, of being able to tolerate noise.

Another useful feature of neural networks is the ability for the neurons to adapt to new inputs, and after training, new features. Some major differences

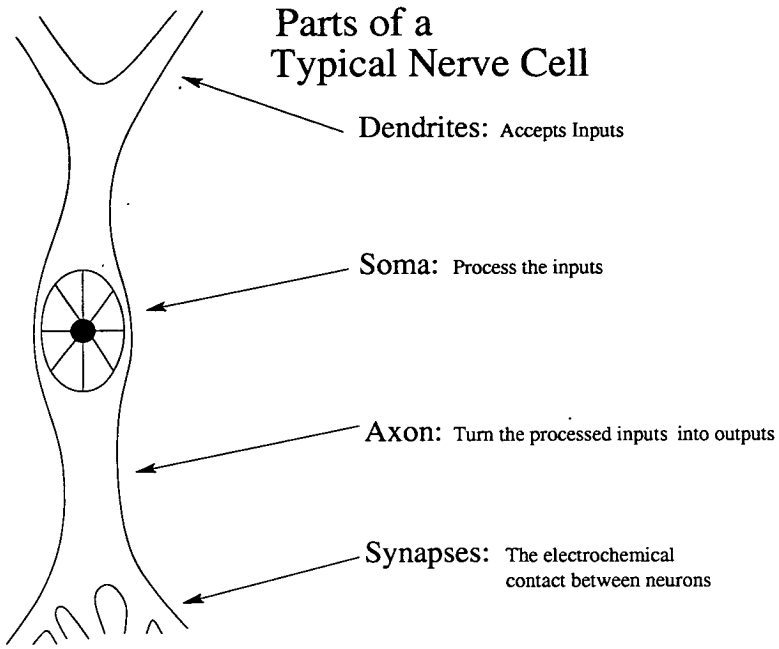

Figure 2.7: A simple biological neuron. This is the basis for an artificial neural network's node.

between biological neural networks (brains) and conventional computers are illustrated by table 2.2.

BNN	Digital Computer
10^{14} separate processors	Few processors
Capable of 10^4 operations per second	capable of up to 100 billion operations per second
Distributed Memory	Centralised Memory core
fairly insensitive to noise in data	Highly sensitive to noise
Ability to learn / adapt	Can only perform exact operations specified to it

Table 2.2: Differences between BNNs (Biological neural networks) and digital computers.

2.5.2 Definitions

Figure 2.8 shows schematically a typical setup of an ANN where squares are referred to as nodes within the network. Each vertical column of nodes in this figure is called a layer. Input data are fed into the "input" layer on the left and the output emerges from the layer on the right. In this figure there is an intermediate layer known as a "hidden" layer. Connections exist between the

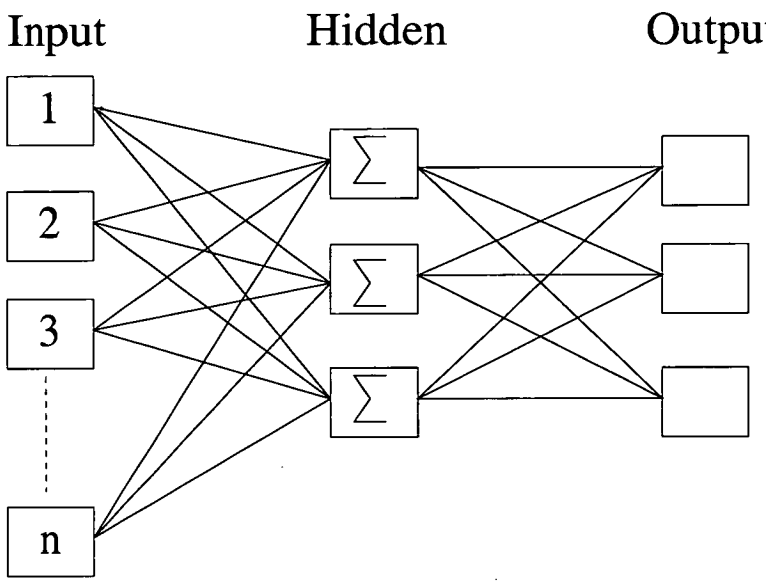

Figure 2.8: A sample neural network consisting of n input nodes, three hidden nodes and three output nodes.

nodes in the input and hidden layer and between the nodes in the hidden and output layer, represented by lines. Other configurations are possible.

Unfortunately there is no set standard notation or definitions in the field of neural networks. For example diagram 2.8 may be referred to by some as being a 1 layered network (excluding both the input and output layer), while others would refer to it as a 2 or 3 layered network. While it is normally clear what is meant when accompanied by a diagram, often diagrams are omitted to save space (especially in articles), leaving the reader to figure out what is going on on their own. This is only one example of the confusions that can arise; there are many others.

In this thesis, figure 2.8 is referred to as a 3 layered network with n input nodes, 1 hidden layer of 3 nodes and 3 output nodes. If a network with another hidden layer is used, the network would be referred to as a 4 layered network with n inputs, p and q hidden nodes, and r output nodes.

Throughout, a neural network, or NN, will refer to artificial neural networks. The words neuron and node will also be used interchangeably.

It should also be noted that there are several opinions on the history of neural networks. The version of neural network history that makes the most sense to me is presented here, but other people argue about what was invented by whom.

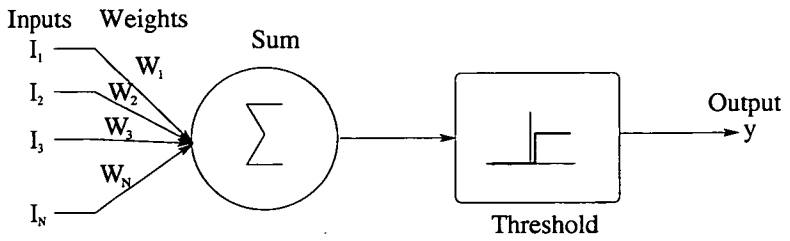

Figure 2.9: The McCulluch-Pitts neuron. This is the basis for modern ANN implementations. The neuron takes in N inputs, multiplies them by weights and sums them. If this sum is greater than a threshold value, the neuron outputs a 1, otherwise it outputs a 0.

These facts should be taken into account when reading other articles about the history of neural networks.

2.5.3 ADALINE and MADALINE

In this section, the simplest neural networks - ADALINEs and MADALINEs - are looked at. These were originally created in the 1940's and are considered the forerunners of today's neural networks. When they were first created and demonstrated, everyone was impressed by their flexibility but this enthusiasm soon waned as people realised that they were only capable of the simplest of tasks.

In the early 1940's, McCulluch and Pitt presented a paper in which they suggested a simple computational model of a neuron (McCulluch and Pitts (1943)). They called this model ADALINE - ADaptive LInear NEuron. Its basic structure is shown qualitatively by figure 2.9.

Here, the neuron takes in N inputs, multiplies them by weights, W_n, and sums them. If this sum is greater than a threshold value, T, then the neuron outputs a 1, otherwise it outputs a 0. Although very simple, this unit could perform several operations, for example, AND and OR operations. Initially weights had to be predetermined and assigned before the unit was run. Later, it was suggested that the weights of the network could be determined without human intervention using training.

Training a network involves changing its weights in order to minimise the error on the output. To do this, a set of example input/output profiles is constructed, called a training set. This should cover the largest range of inputs and outputs

possible. One profile is then selected at random from the training set and run through the network, producing the actual network output, O. The weights of the system are then updated using a simple rule, described by equations 2.20 and 2.21, where η is the learning rate, E is the error on the output before thresholding, and d is the desired output.

Once the weights are updated, a new training profile is selected at random and the network runs through the training procedure again. Once the network is in the right state for all the training cases, the training is stopped.

$$W_i(new) = W_i(old) + \eta E x_i \tag{2.20}$$

$$E = (d - O)^2 = (d - \sum_i W_i X_i)^2 \tag{2.21}$$

The next logical step was to combine several ADALINE units together in parallel, to form a MADALINE - Many ADaptive LInear NEurons (e.g. Widrow and Lehr (1990)). In this case, a number of ADALINE units are all given the same input data and each produces an output value independently. There is a final neuron whose inputs consist of the outputs from these neurons and which performs a majority vote. If over half the networks claim the output is a 1, this unit will output a 1 otherwise it outputs a 0.

Training in a MADALINE is a little more complicated than in an ADALINE. Again, a training set is created and a profile is randomly chosen from this and run through each unit but now all the units compete with each other to decide which unit gets updated. Each unit calculates its error according to equation 2.21, as before.

The winner of this competition is the unit with its error closest to 0, but that is outputting the wrong value. Only this node's weights are updated using equation 2.20.

Although this model was highly original, it was also highly limited. It could only solve certain classes of problems, known as linearly separable problems.

The problem of linear separability can be illustrated with a simple example of a ADALINE / MADALINE system with two inputs in two cases - the AND problem and the exclusive-OR (XOR) problem. Here, the input space will be 2-dimensional[2]. If the system is attempting to perform an AND operation on its

[2]Provided there are only two inputs

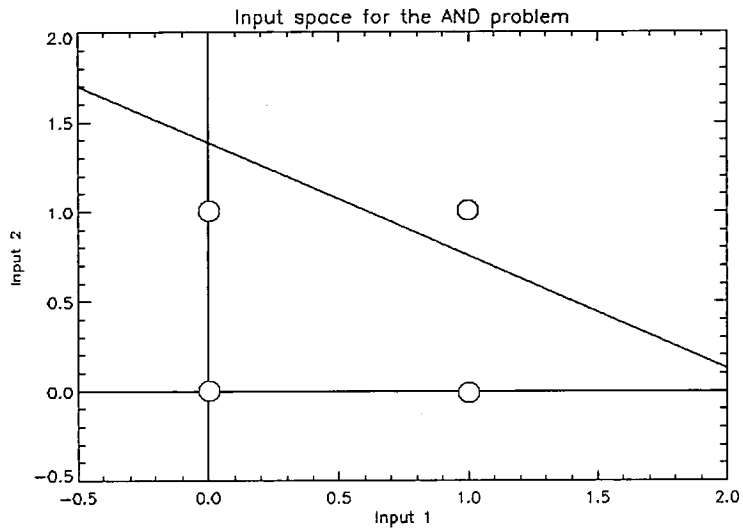

Figure 2.10: Input space for a 2-input AND problem. The aim of a neural network in this case is to find a threshold (line in the diagram) that separates the red dots (the neuron should output a 0) from the green dot (the neuron should output a 1).

input data, the input space will look like figure 2.10, where a red point indicates the output neuron should not fire, while a green point indicates that it should. Here, the system is attempting to create a line in input space that separates the red and green dots. Then, when the point representing inputs (I_1, I_2) is plotted, if it is above the line, the output of the system will be one value (In this case 1) and if it is below then the output will be another (0).

In the AND case, this is relatively simple - only one point needs to be above the line. This is called a "linearly separable" problem[3]. The problems with ADALINEs / MADALINEs (also no-hidden layer perceptrons - see section 2.5.4) comes when the problem is not linearly separable. A simple example of this is the XOR problem, as represented in figure 2.11. As there is no straight line that can be drawn which separates the red and green dots into two distinct groups, this is not solvable by an ADALINE system. This problem was illustrated by Minsky and Papert (1969).

The other problem with ADALINE / MADALINE systems was that the output could only be binary. This was solved by the introduction of perceptrons in the late 1950's.

[3]i.e. the solutions can be separated by a linear line

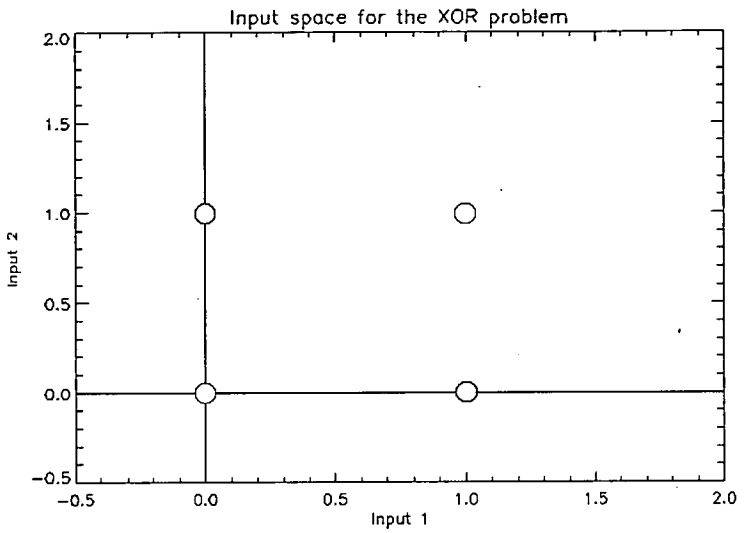

Figure 2.11: Input space for a 2-input XOR problem. Here, no threshold (line) can be drawn that separates the red dots (neuron output of 0) and the green dots (neuron output of 1) thus the problem cannot be solved using an ADALINE neural network.

2.5.4 Perceptrons

In the late 1950's Rosenblatt (1958) suggested an improvement to the McCulluch-Pitt neuron in order to make the output continuously valued. A network of these new neurons was called a perceptron.

Basic perceptrons are made up of two layers of nodes - an input layer and an output layer. The input nodes each take one input and pass that value to every output node. The output layer is made up of a number of these new nodes. Each output node has a number of inputs from the input layer. This is shown in figure 2.12.

These new neurons differ from the McCulluch-Pitt model in one important way. In a McCulluch-Pitt neuron, the output can either be a 1 or a 0, but in these neurons, the output is continuously valued. This is done by changing the activation function from a threshold function to a continuous function. The most commonly used activation function is given by equations 2.22 - 2.23, where ω_i is a weight, which may be any real number, $V(\sigma)$ is the output from the node and I_i is an input. This is known as the logistic sigmoid activation function.

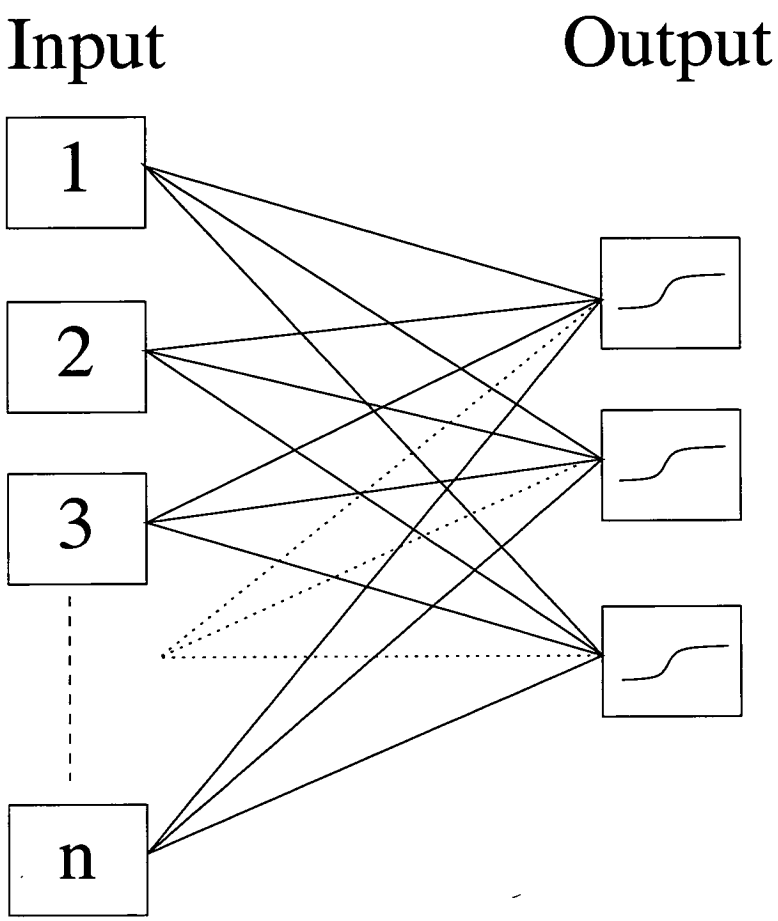

Figure 2.12: A Simple perceptron. The network consists of two layers of nodes connected by a series of weights. In the input layer, each node receives one input and passes this to each output. The output layer consists of a number of nodes that perform an activation on their inputs. This is an improvement on the MADALINEs introduced previously as the outputs are now continuous.

$$V(\sigma) = \frac{1}{1 + exp(-\sigma)} \qquad (2.22)$$
$$\sigma = \sum_i \omega_i I_i$$

This is used as its derivative is easily replaced by functions of $V(\sigma)$ (equation 2.23), allowing easy implementation into simulations as $V(\sigma)$ is the value outputted by the node. This property is useful during training.

$$\frac{dV(\sigma)}{d\sigma} = \frac{exp(-\sigma)}{\{1 + exp(-\sigma)\}^2} \equiv V(\sigma)\{1 - V(\sigma)\} \qquad (2.23)$$

Training a network with no hidden layer is analogous to training on a MADALINE. The difference is that the continuous nature of the activation function introduces a derivative term into the training rule. Updating the system's weights is then done using equation 2.24 where η is the "learning rate", which must be specified prior to training and will absorb any numerical components of the derivative into its definition. The learning rate is used to control how much the weights change during an update cycle. A value of 1.0 will result in the weights being updated fully to accurately reproduce the particular training profile. This is undesirable as, during each update in the training cycle, the network will forget all it has learnt before.

$$E_i = (d_i - O_i)^2$$
$$\omega_i(New) = \omega_i(Old) + \eta E_i I_i \frac{dV(\sigma)}{d\sigma} \qquad (2.24)$$

Here E_i is the error on node i, d_i is the desired output from node i (in the training and O_i is the true output. Other quantities are as in section 2.5.2. Further discussion about perceptrons can be found in appendix A.

When Minsky and Papert pointed out the problems with neural networks and linear separability, the neural network community was quick to respond by suggesting improvements to remove the problem by adding new hidden layers to the perceptron (e.g. Rummelhart et al. (1986)). These hidden layers comprise of neurons with a continuous activation function and go between the input and

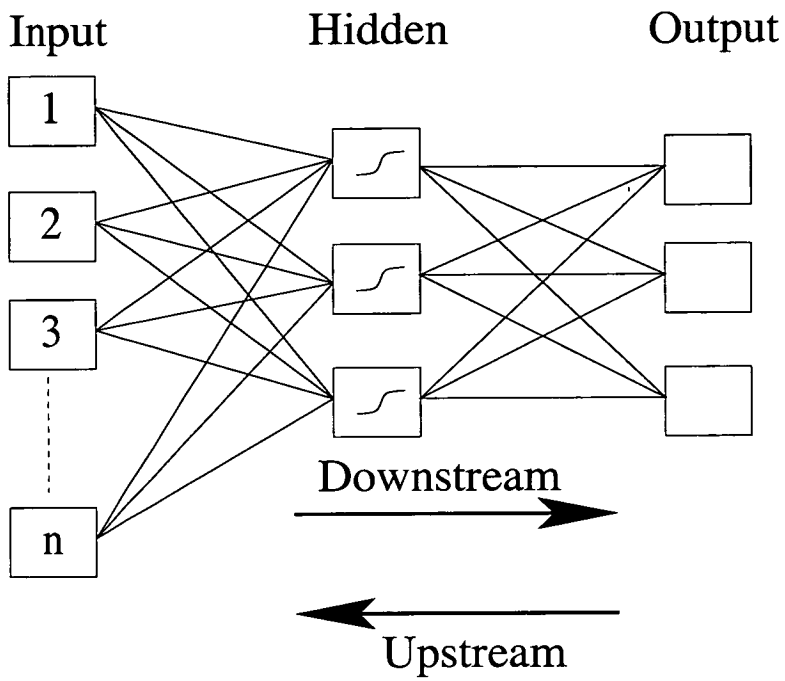

Figure 2.13: A neural network with a hidden layer of three nodes. These hidden nodes perform a non-linear activation function to allow continuous outputs. The arrows show the relative directions of upstream and downstream as used in the text.

output layers. The network topology is now shown in figure 2.13, with one hidden layer.

Adding these hidden layers solves the linear separability problem. Having extra hidden layers corresponds to being able to add extra lines to the input space diagrams (figures 2.10 and 2.11 of section 2.5.3), so the XOR problem could be solved as in diagram 2.14, where between the lines, the networks output one value (1.0) and outside the gap it outputs another (0.0)[4] (Russell (1993)).

Unfortunately, solving the linear separable problem created its own problem - how to assign "blame" to the input-to-hidden weights. For example, if there is an extra layer in a network, when the output and error is calculated for a given input, how do you know which weight is responsible for each proportion of the error?

This was a problem that stopped neural network research in its path for over 10 years during which time, excitement about neural networks vanished, and

[4]Assuming McCulluch-Pitt neurons

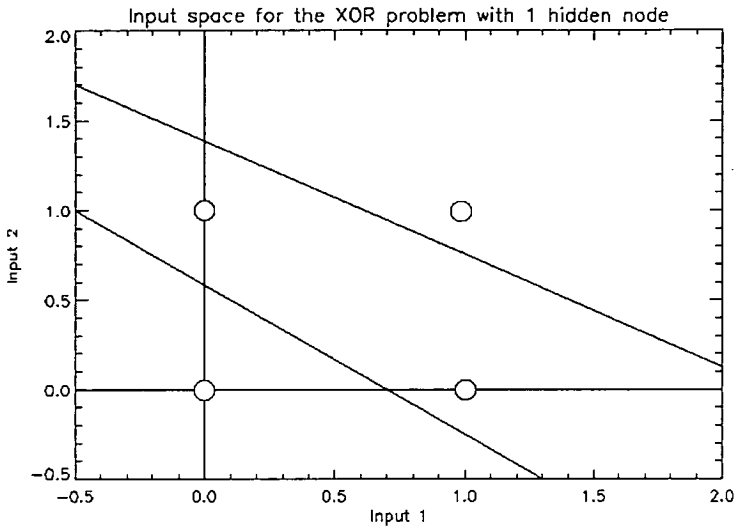

Figure 2.14: Input space for a 2-input XOR problem using 1 hidden node. Using hidden nodes in a neural network is equivalent to allowing additional lines to be drawn in input space, allowing the XOR problem to be solved.

funds for research dwindled. This changed in 1986 when Rummelhart et al. (1986) popularised a method for training perceptrons with hidden layers[5] called backprop.

2.5.5 Back-propagation

Back-propagation, or backprop, or back-propagation of error, is a method of training a perceptron with hidden layers using two stages.

The first stage is similar to what has gone before: the network is fed the input for a randomly selected training profile and its output is calculated. This output is then be used to calculate the error for each output node, $E_i = (d_i - O_i)^2$, as before.

The true power of back-propagation lies in what happens next, the so-called "Back-pass".

The basic idea of this pass is that the error is allowed to propagate upstream (i.e. to previous layers), where each node calculates its contribution to the overall error, stores this and removes this contribution from the error, before passing the

[5]This method was independently discovered in Paul Werbos in 1975 and Rumelhart *et. al.* in 1986

new value of the error upstream again.

When the error, E_i, starts at the output layer, it is necessary to calculate δ_i, a "sensitivity" factor for the error. This is defined as equation 2.25 where $\frac{dV(\sigma)}{d\sigma}$ is the derivative defined as equation 2.23 previously.

$$\delta_i = \frac{dV(\sigma)}{d\sigma} E_i \tag{2.25}$$

Having found δ's for all the output nodes, it is then possible to propagate these upstream to find the values at the next layer, using equation 2.26, where the summation is over all nodes downstream that are connected to node k.

$$\delta_k = \frac{dV(\sigma)}{d\sigma} \sum_h \omega_{kh}\delta_h \tag{2.26}$$

This is calculated iteratively upstream until all nodes in the network have δ values. Having successfully given every node in the network a δ value, the weights are updated using the "delta training rule", as given by equation 2.27 (Derivation is given in Braspenning et al. (1995)). It should be noted here that the subscript k refers to the source node of the link (i.e. the upstream node), while h refers to the downstream node. A more detailed explanation of how backprop operates in a multi-layered perceptron can be found in appendix A.

$$\omega_{kh}(new) = \omega_{kh}(old) + \eta\delta_h V_k(\sigma) \tag{2.27}$$

This is the basic building block of most modern neural networks. Many additions have been proposed to this delta rule, but only two of them have really been embedded into the foundations. This is the addition of the momentum term (Hertz et al. (1991)) and the inclusion of a weight decay term (Krogh and Hertz (1992)), producing what is known as the "Generalised Delta Rule". This includes the addition of a fraction of the previous weight change ($\Delta\omega_{kh}(old)$) for a momentum term and a fraction of all the weights in the current layer for a decay term, and is given by equation 2.28. This introduces two new terms, α and ν, the momentum and weight decay coefficients respectively, which must be chosen before training the network.

The purpose of both these are different. The momentum term allows the weight change to build up, and get over any small bumps in weight-space. The decay term favours smaller weights in the system and helps prevent over-fitting

(see section 2.5.5).

$$\omega_{kh}(new) = \omega_{kh}(old) + \eta\delta_h V_k(\sigma) + \alpha\Delta\omega_{kh}(old) - \nu\sum_m \omega_{mh} \tag{2.28}$$

This is still the most common basis for most of today's neural networks. Many implement their own special modifications to this algorithm but these tend to be extensions of the backprop algorithm.

Special Types of Nodes

In all that has proceeded, it has been assumed the topology of a network has consisted of a series of layers of nodes where each node is connected to all the nodes of the previous layer upstream and every node in the next layer downstream, that is to say there have been no *skip layer connections* (e.g. Ripley (1997)). These are relatively simple in theory. The main idea is to have a node that links not to every node in the next layer, but to "skip" a layer or layers. This is sometimes used if there is a known (or suspected) linear relation between an input and output. In this case, a skip layer connection may be connected from one or more input nodes straight to one or more output nodes, as well as through the hidden layers. If there is a linear relation, the skip-layer weights will become much greater than the hidden-to-output weights and so will dominate the output.

The other type of special node is called a bias (e.g. Haykin (1998)). This is a type of node that is implemented in nearly all neural networks, but generally not discussed in literature. This node is a node whose output is always 1, and is generally connected to every hidden and output node. A sample network with a bias included is shown in figure 2.15. This acts to move the activation function as shown in figure 2.16. Similar results are found for other perceptron activation functions. A 'bias of 2.0' here implies the bias-to-node weight has a value of 2.0 (achieved by using setting the weight to 2.0).

Stopping Training

One major problem of using backprop is knowing when to stop training the network. The network could be trained indefinitely, constantly reducing the

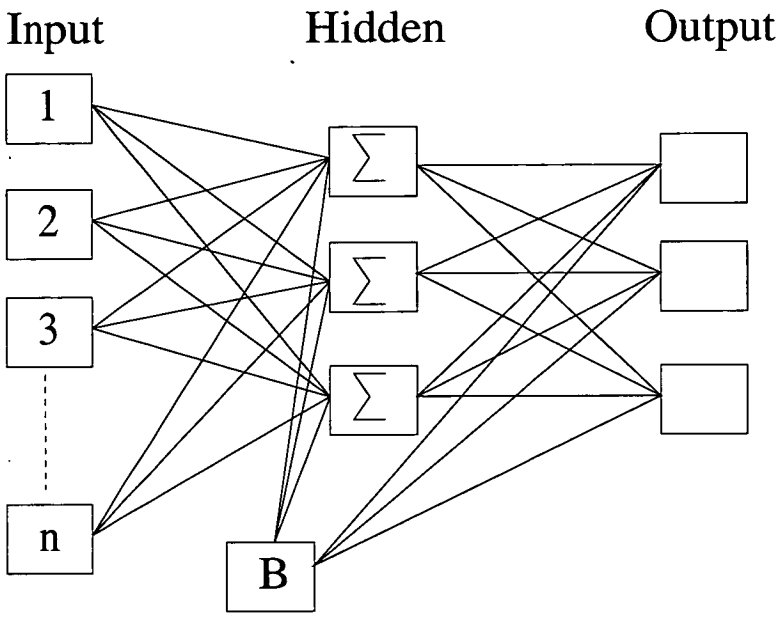

Figure 2.15: A simple network with a bias node, B, includes. Traditionally, bias nodes are included in almost every neural network, but are not discussed for reasons of brevity.

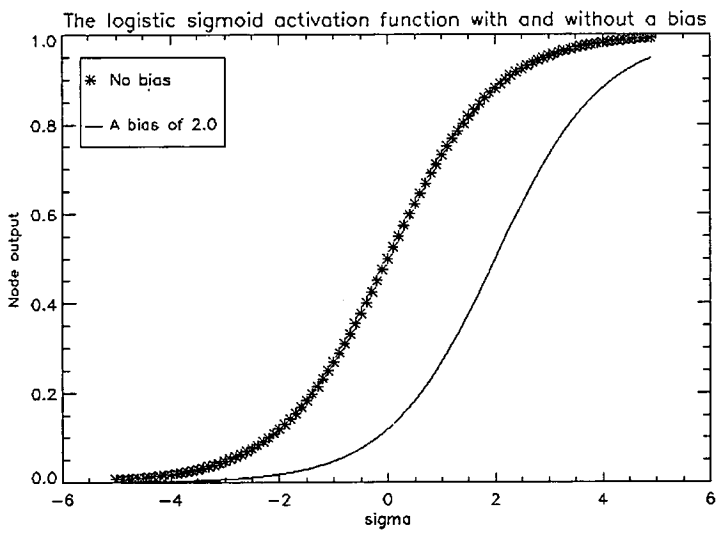

Figure 2.16: How the sigmoid activation function varies with a bias node. With a bias of 2.0 included, the summed inputs (σ) must be increased to get the same activation level.

error. However, this may lead to over-fitting if the network is too complex. This is the phenomena of the network learning all the noise in the data which leads to poor generalisation. However, it is in general not possible to know if a network is too complex for a given task until the network is fully trained and tested.

The most common, and simplest, way to combat this is to have a small set of data to check how the network is getting on. This is the validation set. At regular intervals during training, the network is tested (but not trained) on the validation data. When the error on the validation set either becomes sufficiently low (if asked to work out a problem to given precision), or the validation error starts to rise, the training is stopped, and the weights are restored to the values that gave the lowest validation error. This has some minor problems of its own. During training, the error may well go up and down by quite a large amount and the question arises "How do you define when the validation error is going up due to over-fitting and not just through training?". One way around this is to store the best validation error. When the current validation error is better, it is stored. If it goes up some fraction (usually 1.2x to 2.0x) of the best validation error this is an indication to stop.

Choosing Node Numbers

Another problem with any type of neural network is how to choose the number of hidden nodes and the number of hidden layers (the number of input and output nodes is fixed by the given number of inputs / outputs required by the system). It has been shown by Hornik et al. (1989) that only one hidden layer is ever required, but this could lead to a network having 1000's of hidden nodes in one layer, whereas it may be faster to have several layers of 100 each.

The question of how many hidden nodes to put in can be resolved in one of three ways: guessing, pruning or growing.

Guessing, though it sounds primitive, is still the most popular way due to some technical difficulties associated with the other methods. This involves putting in a number of nodes, training the network, see how it performs then adding or subtracting more units to see how that affects the system.

Pruning, or "brain-damaging" is a another common way of getting hidden node numbers (Romaniuk (1993)). Here, a network starts with a lot of hidden nodes. Training proceeds for a set number of epochs after which time, the

weights are reviewed. The hidden-to-output weight(s) that are closest to zero get removed, along with the source node (i.e. the node that the weight originated from). This then requires the network to be retrained. The procedure continues until all node weights are far enough away from zero.

The final way the number of nodes is commonly chosen is by growing (e.g. Sakar and Mammone (1993)). This is similar in style to pruning. The network starts with a small number of nodes (typically 0). Training proceeds for a set number of epochs after which time, the validation error is calculated. If this is above a predefined limit, some new nodes are added. This is repeated until either the validation error drops below some level, or a maximum nodes are added (determined by the user). This method is very prone to over-fitting but does train faster than a pruning network.

All these methods require some user input. Pruning requires a starting number of nodes and growing requires the maximum size of the network, so it doesn't really eliminate the problem of choosing the number of hidden nodes, it just rephrases it.

Clearly ADALINE and MADALINE systems will not be appropriate for an EOS-MLS forward model, as their outputs are limited to binary, but of the schemes discussed here, a multi-layered perceptron, trained with backprop, looks promising. This scheme will be investigated in chapter 3 when using a much simplified forward model and chapter 5 with a more realistic forward model. Chapter 4 also looks at using a multi-layered perceptron with backprop to retrieve tangent pressures for use in an assimilation scheme.

2.6 Previous Work

This section aims to give an overview of some of the work carried out on the use of neural networks in an atmospheric science context. There have been many uses of neural networks in retrieval situations (for example, Jiménez (2003)) that have worked in the past. In these cases, the neural network inputs are the radiances returned from instruments (typically satellites) and the outputs have been atmospheric profiles. In particular, Jiménez (2003) used a neural network to retrieve several chemical species from Odin-SMR measurements (Murtagh et al. (2002)). A multi-layered perceptron was trained using Bayesian techniques (not covered

here - see MacKay (1995)), that used radiances from several bands on the instrument to retrieve O_3 and ClO profiles (among others) that were compared to profiles retrieved using standard retrieval techniques.

It was found that, using model simulations of the Odin-SMR measurements, the neural network gave comparable errors when compared to traditional retrieval techniques but was much faster. When using real measurements, it was found that there were biases of up to 10% at some heights in the neural network retrieved profiles. This was determined to be due to limitations on the training data as opposed to a problem with the technique. Overall, it was found that neural networks provide "a very attractive alternative to the operational inversions".

There have been other attempts at using neural networks in a variety of ways within data assimilation. One of these attempts involved using neural networks to compute the atmospheric fluxes and cooling rates within a 4D-variational assimilation scheme at the ECMWF (Chevallier and Hahfouf (2001)). In this case, it was found that although the atmospheric fluxes and cooling rates could be calculated, the neural network Jacobians had some irregularities but in general "[...] this approach is able to provide fast computations with good accuracy".

There has been only one serious attempt to create a forward model based on neural networks that I could find. This was done by Krasnapolsky (1997). Using a neural network, he developed a forward model, called the OMBFM1, for the SSM/I instrument (Hollinger et al. (1990)) across 5 of its channels. It was found that the OMBFM1 could produce brightness temperatures in all channels, together with associated derivatives, to an acceptable standard. It was also found that the neural network based forward model was much simpler than a traditional forward model to run in a retrieval scheme.

The task of creating a neural network forward model for the EOS-MLS is more difficult than for the SSM/I instrument as the EOS-MLS is a limb sounder. The OMBFM1 takes 4 inputs: wind speed, columnar water vapour, columnar liquid water and sea surface temperature and produces 5 outputs: the brightness temperatures in the 5 channels being modelled. In contrast, EOS-MLS radiances are affected by temperature variations throughout the atmosphere as well as other chemical species (see section 2.3.1), which drastically increases the number of inputs required by the system. In addition, the EOS-MLS produces 120 outputs per channel per scan resulting in many more outputs. There are further compli-

cations when determining pointing geometry which do not affect nadir sounding instruments (chapter 4). These difficulties mean any forward model must perform additional steps when calculating radiances for limb sounders as opposed to nadir sounders. In a neural network, this is represented by the need for additional hidden nodes to deal with the added complexity.

Chapter 3

Preliminary Evaluation of the Neural Network Forward Model

3.1 Introduction

In order to ensure that neural networks are a feasible method of producing a forward model, a much simplified initial experiment was conducted, involving limiting the radiances produced.

This chapter gives an overview of how the neural network forward model was created, the training method employed and details of the first experiments in modelling a forward model in limited circumstances.

3.2 The First Model

Only one channel in one band was modelled. It was decided to model only band 1 in this way. Band 1 was used as it is centered on an oxygen line and operates as a single sideband. As the concentration of oxygen in the atmosphere is effectively constant and the line the band is centered on (118.75 GHz) is strong and isolated this means good radiances can be produced using temperature as the only species input. In order to simplify the problem further, the tangent pressure levels of the radiances were assumed to be constant across profiles (see section 2.3.1). This means that the each minor frame radiance is measured at the same pressure in each major frame.

3.3 The Network Architecture

This section deals with the construction of the neural network. It explains the architecture as well as the normalisations and transfer functions.

As explained in chapter 2, a neural network is a quick way of estimating functions using nonlinear neurons connected in a network. In order to keep the number of outputs reasonable, it was decided to use one neural network per channel for the EOS-MLS, rather than using one neural network for all channels in the band. This means that there are 120 outputs for the network. In addition to this, there are 73 inputs, the temperature profile.

The network used is a multi-layered perceptron. The number of hidden nodes within the network can be varied, along with the number of hidden layers. The transfer function (the activation on the hidden and output nodes) can be varied to be either a sigmoid function (equation 3.1), a hyperbolic tangent (equation 3.2) or a linear transfer function (equation 3.3) where the symbols are as used in section 2.5.4.

$$V(\sigma) = \frac{1}{1 + exp(-\sigma)} \qquad (3.1)$$
$$\sigma = \sum_i \omega_i x_i$$

$$V(\sigma) = \tanh(\sigma) \qquad (3.2)$$
$$\sigma = \sum_i \omega_i x_i$$

$$V(\sigma) = \sigma \qquad (3.3)$$
$$\sigma = \sum_i \omega_i x_i$$

Because the nonlinear transfer functions (described above), only return values in the range [0,1] or [-1,1], some normalisation must be applied to the outputs of the system. In addition, some form of normalisations is also applied to the inputs

of the system, but for different reasons.

As a typical temperature profile covers a large range of values, if these values were fed directly into a neural network, several problems would arise. First, during training, the values combined with typical initial weights would result in the hidden nodes being saturated. This means all of them would output a constant value of 1 (or very nearly). In this case, it is not possible to train the network very efficiently, because the training relies on the gradient of the transfer function. If the node becomes saturated (either too high or too low), this gradient falls to zero. Although this effect can be mitigated by using much lower initial weights, these values would have to be constrained to be at most given by equation 3.4, where W_i is the initial weight, N_I is the number of inputs and $\bar{I}$ is the mean value of the inputs.

$$W_i \approx \frac{1}{N_I * \bar{I} * 2} \approx \frac{1}{73 * 200 * 2} \approx \pm 0.00003 \tag{3.4}$$

This means that the initial weights would have to be in the range $[-0.00003, 0.00003]$ to avoid saturating the hidden nodes. If the weights get smaller, the internal calculations within the neural network will start losing precision. A further problem arises when more inputs are needed (see chapter 5) and this initial weights range would have to be reduced further.

Another problem with unnormalised inputs is that some inputs may have a larger influence on the outputs than others. Although this is to be expected given the problem, there might be a single input that has a very large range of values while all the rest have a much smaller range. In this case, the much larger changes in one input may mask the changes in other inputs which could be more important. Although this effect should dampen with training, it may increase training time dramatically, or potentially not manifest itself until training is complete.

For these reasons, both the inputs and the outputs must be normalised. In order to normalise the inputs, equation 3.5 is used, where $I_n(i)$ and $I_u(i)$ are the normalised and original un-normalised input to node i respectively.

$$I_n(i) = \frac{I_u(i) - \min^*(I_u(i))}{\max^*(I_u(i)) - \min^*(I_u(i))} \tag{3.5}$$

To allow some growth outside normal ranges, the maximum and minimum

factors in equation 3.5 are scaled by 10%. When the maximum or minimum value is greater than zero, $\max^*(I_u(i)) = 1.1 * \max(I_u(i))$ and $\min^*(I_u(i)) = 1.1 * \min(I_u(i))$. When the maximum or minimum value is less than zero, $\max^*(I_u(i)) = \max(I_u(i))/1.1$ and $\min^*(I_u(i)) = \min(I_u(i))/1.1$.

These input normalisations result in the network inputs being approximately in the range $[0, 1]$[1]. This allows the initial weights to be in the approximate range $[-0.01, 0.01]$, which allows more accurate internal calculations and prevents the hidden nodes being saturated. Normalising the inputs also has an advantage in equalising the input's influence on the outputs.

The raw outputs from the network will be in the range of $[0, 1]$ or $[-1, 1]$ depending on which of equation 3.1 or 3.2 are used. These raw values are then converted to radiances using equation 3.6 in the case of a sigmoid transfer function or equation 3.7 in the case of a hyperbolic tangent transfer function. When a linear transfer function is used, no output normalisations are applied.

$$O_u(i) = O_n(i) * \{\max^*(O_u(i)) - \min^*(O_u(i))\} + \min^*(O_u(i)) \tag{3.6}$$

$$O_u(i) = \gamma(i) * \{\max^*(O_u(i)) - \min^*(O_u(i))\} + \min^*(O_u(i)) \tag{3.7}$$
$$\gamma(i) = \frac{O_n(i)+1}{2}$$

Here, the notation is similar to that for the input normalisation; $O_u(i)$ is the unnormalised (true) radiance for network output i, $O_n(i)$ is the network output from output node i, $\max^*(O_u(i))$ and $\min^*(O_u(i))$ are the maximum and minimum true radiances in the training set, scaled to allow for larger / smaller values in the same way as the inputs scaling. Figure 3.1 shows the final network architecture.

[1]Assuming the absolute maximum and minimum values are approximately 110% of the training set values

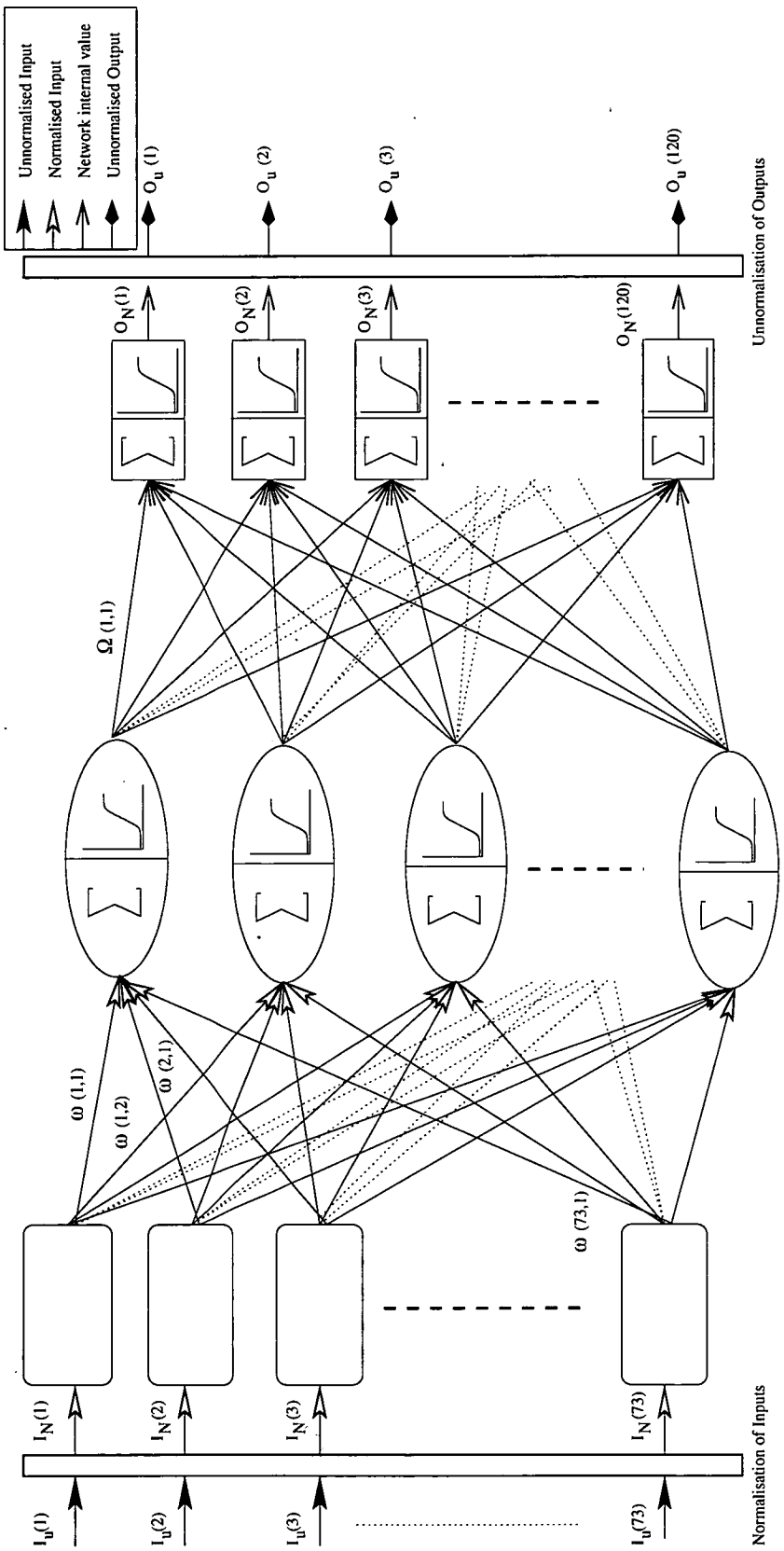

Figure 3.1: The architecture of the network. The inputs are the (unnormalised) temperature profile. These are first normalised and passed to the input layer. The input layer does nothing to these values, except to pass them to all the hidden nodes. The hidden nodes then sum the weighted inputs and performs an activation, passing the result to the output nodes. The output nodes again perform a summation of each of its weighted inputs and passes a value out. These values are then unnormalised to produce the final radiance. As is conventional, the bias nodes are not shown in this figure.

3.4 Training the Network

In order to be useful, a neural network must be trained. This section describes the training methods employed and how the training data were generated.

As discussed in chapter 2, there are many ways to train a neural network. The network described here was set up to be trained in several different ways. The first way is using backprop (discussed in section 2.5.5). Using backprop, there are several parameters that can be varied - the learning rate (η), the momentum of the system (α) and the weight decay factor (ν).

The second training method employed is a technique called quickprop. Quickprop is a technique developed by Fahlman (1988) as an alternative to the backprop algorithm. The idea is to use a copy of the derivative of the error with respect to each weight for the previous training profile and use this combined with the derivative from the current training profile to calculate an approximation to the second derivative. The weights are then updated according to the rule given by equation 3.8.

$$\Delta\omega(t) = \frac{S(t)}{S(t-1) - S(t)}\Delta\omega(t-1) \tag{3.8}$$

$$S(t) = \frac{\delta E}{\delta \omega} = \gamma_h V_k(\sigma) \tag{3.9}$$

Here, $\Delta\omega(t)$ is the change to be made to the weight at timestep t and $S(t)$ is the current value of the derivative of the error with respect to the weights, $\delta E/\delta\omega$, calculated using equation 3.9 where γ_h is the "sensitivity factor" defined as equations 2.25 - 2.26 in section 2.5.5 of chapter 2[2]. As before, $V_k(\sigma)$ is the value generated by the node k. Putting this update formula into the form of the backprop equation (equation 2.28) gives an update rule of equation 3.10. As before, η is the learning rate. Unlike backprop, quickprop has no momentum or weight decay terms as it doesn't suffer from the problems that they were intended to solve.

$$\omega_{kh}(new) = \omega_{kh}(old) + \eta\frac{S(t)}{S(t-1) - S(t)}\Delta\omega(t-1) \tag{3.10}$$

One problem that sometimes occurs with quickprop is that the denominator

[2] γ is used in place of δ as the sensitivity here to avoid confusion

of equation 3.10 becomes so small that a floating point overflow occurs, resulting in infinite weight changes. The normal way of dealing with this is to introduce a "maximum growth factor", μ (Fahlman (1988)). Using this, the absolute value of the weight change is not allowed to be more than μ times the previous weight change. If the weight change is greater than this threshold value, a constant value is used in its place.

Quickprop is also based on several assumptions. The first assumption is that the weight space is approximately parabolic in shape. This means that there is a well defined minimum that can be reached, and that weight space is relatively smooth. In many cases, this is a reasonable assumption. If weight space isn't parabolic, the method will still converge on a solution but will tend to become stuck at local minima instead of the global minimum.

The second assumption is that the slope of the error vs. weight curve for each weight is not greatly affected by other weights that are changed at the same time. This is generally a good assumption when each output node is affected in different ways by the input nodes. In this case, the hidden nodes will act to separate out these effects. When creating the forward model for the EOS-MLS though, each minor frame output[3] is dependent on similar inputs as those around it as well as any temperature inputs from the atmosphere above it due to the viewing geometry. This means that this assumption could be risky in this case.

3.4.1 The Training Set

The training data were generated using a full 2-D (i.e. the atmosphere is assumed to be horizontally homogeneous) forward model for the EOS-MLS created by H. Pumphrey, which has been shown (Pumphrey (2006)) to reproduce the "official" EOS-MLS forward model, described in Read et al. (2004), to within ≈ 1 K. Here, the model of Pumphrey is assumed to be accurate. The training set was generated at constant tangent pressure levels with only temperature data included. The training set contains 2,000 radiance profiles (along with corresponding temperature profiles). Of these, 1,500 are used for training, 300 for validation (used to measure training progress) and 200 for testing (used at the end of training to ensure the network is well-trained). These values were chosen so that each

[3]Described in section 2.3.1 of chapter 2

set covers at least one orbit of the EOS-MLS, to give maximum coverage of the expected temperature profiles, while allowing enough profiles remaining from a full day's data to allow further testing of the network.

In order to ensure that the training set is representative of the complete input / output space, the following tests were undertaken. First, several parameters, such as the mean, standard deviation, maximum and minimum values for each minor frame within the profile were compared.

The second test is slightly more complex. This process gives a qualitative view of which areas of input/output space are well represented and can be created as follows. First, each height within the entire set is binned into a set of 10 ranges. This procedure is then done again with the training set and the same ranges as the complete set. These ranges are then normalised and the values from the training set are subtracted from the values of the complete set, producing a difference between the two sets.

Within a well represented area, the final ranges should have a value near zero. A threshold value of 20% was chosen as the cut-off for defining well-represented areas. Areas of the training set with 20% fewer profiles than the complete set are unrepresentative and need to be improved.

In this case, 3496 profiles are used as the complete set. This represents one day's worth of profiles and is a good representation of profiles across all latitudes the instrument measures at. The training set consists of the 1500 training profiles. The results in the case of temperature profiles are found in figure 3.2. This shows two sub-figures. On the left, is a figure showing the mean, maximum, minimum and standard deviation for each height. The red dashed line is the training set, and the solid black line is the complete set. As can be seen, all of these lines are almost perfectly matched. The figure on the right gives an indication of the coverage as described above. The great majority of the area covered is indeed near the "Perfectly Represented" colour, with a small amount number of underrepresented and overrepresented areas appearing. The extremes of the scale are where the training set has 15% more profiles (over-represented), or 15% fewer profiles (under-represented) within the area, within the threshold defined above.

The same procedure was also carried out to look at the network outputs (i.e. the radiances). This can be found in figure 3.3 and is in the same format as

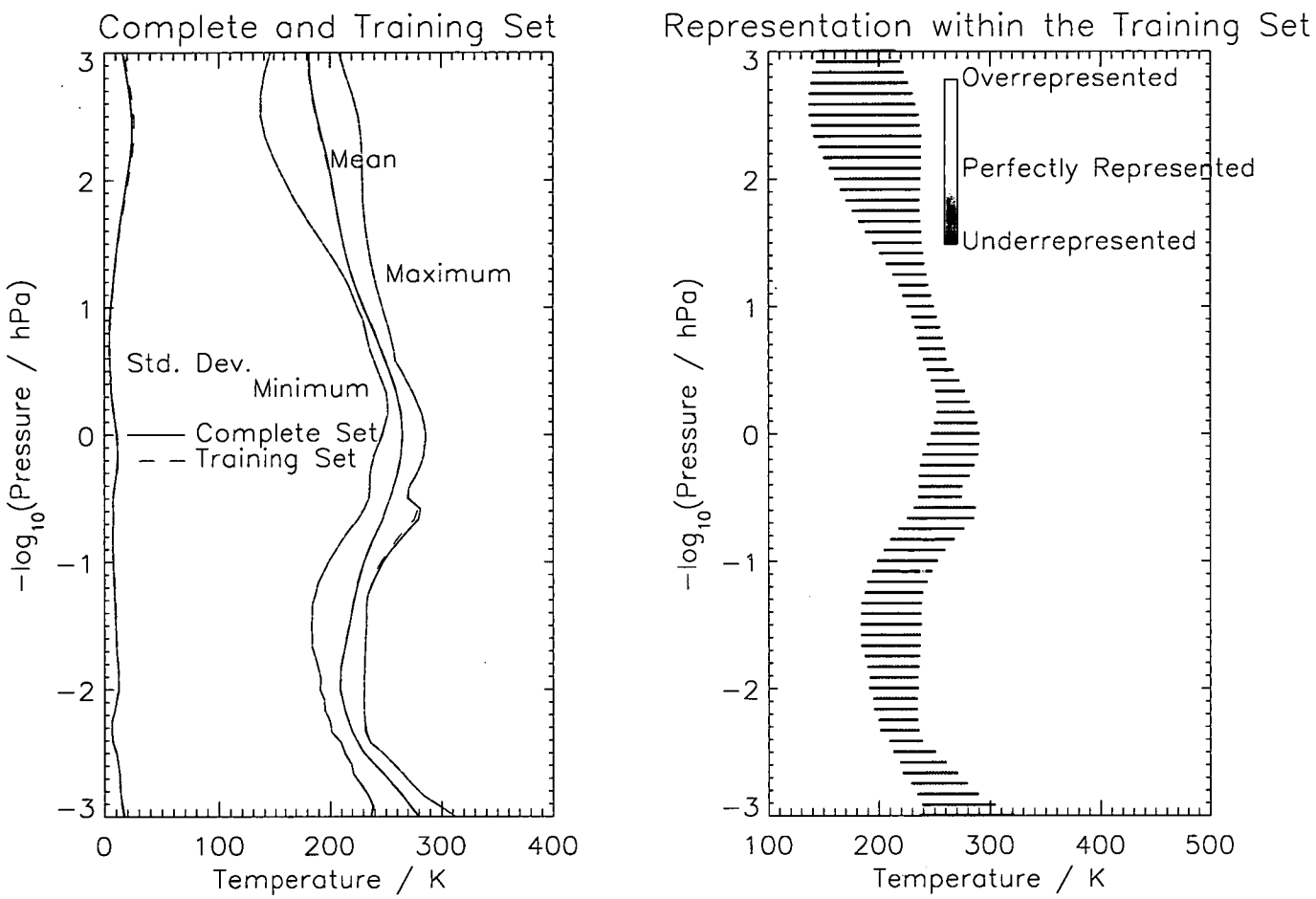

Figure 3.2: A figure showing that the temperature training set is representative of the entire set. The left figure shows the maximum, minimum, mean and standard deviation of the entire set (red dashed lines) and the training set (solid black lines). The right figure gives a qualitative overview of how well different areas are represented with well represented areas displayed in blue-green. A small number of areas (around $y = -0.9$) show deviations away from this well-represented area. Here, the extreme under- and over-representation corresponds to 15% fewer/more profiles in the training set than in the complete set.

figure 3.2. Again, it can be seen that the maximum, minimum, mean and standard deviation are all very similar in the training set and the complete set. The right sub-figure shows that the complete set is well represented by the training set in almost all areas, with no seriously underrepresented areas. Here, the scale used is from 5% under-represented to 5% over-represented, well inside the threshold for well-represented areas. This suggests the training data gives a good representation of the complete set.

The training process consists of showing the network 7500 profiles randomly selected from the training set and updating the weights according to the errors on those profiles. The network is then validated using the 300 validation profiles and if the network error is better than in previous validation runs, the network weights are stored. The training run is then restarted. This process continues until the validation phase does not produce any better weights for the system for 50 validation runs (epochs) in a row. After this, the best weights are restored and the network is run with the 200 testing profiles.

3.5 Results

3.5.1 Initial Trials

This section gives some results of tests carried out using the neural network and includes the effect of training algorithm, choice of transfer function and hidden node numbers.

A number of tests were carried out using a neural network forward model under a range of different circumstances. These include changing the number of hidden nodes, changing the training method and changing the transfer functions used by the nodes within the network.

Results from an individual run will usually be presented here in the form of a four-panel graph. An example graph can be seen in figure 3.4. Here, the upper left panel shows a sample profile from within the testing set, chosen at random, showing the real forward model output (solid line) with the corresponding network forward model outputs at the end of training (crosses). The upper right panel shows the absolute errors on each network output for this profile at the end of training. The lower left panel contains information about the complete test set. This includes the following information:

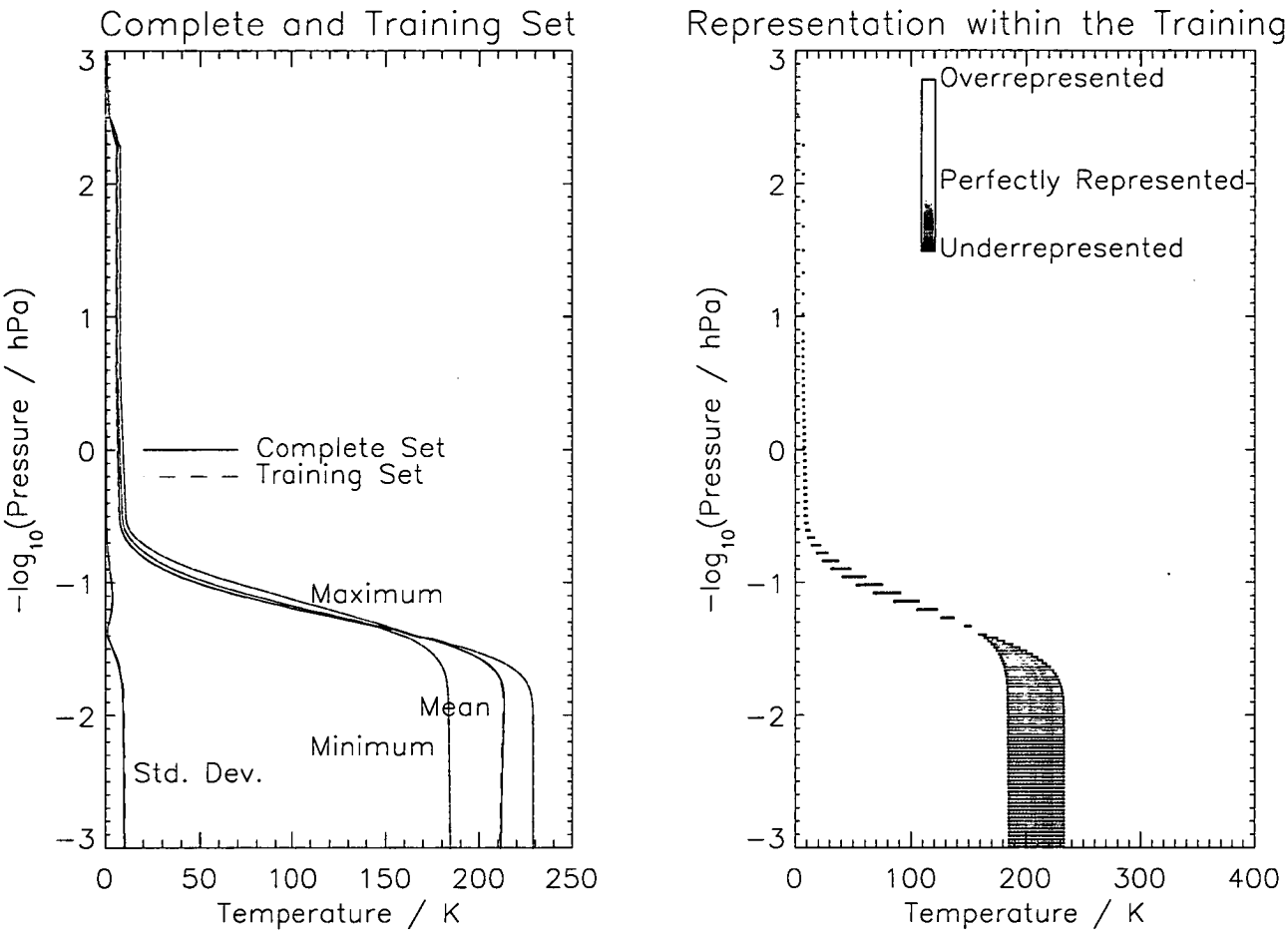

Figure 3.3: A figure showing that the radiance training set is representative of the entire set. The left figure shows the maximum, minimum, mean and standard deviation of the entire set (red dashed lines) and the training set (solid black lines). The right figure gives a qualitative overview of how well different areas are represented, with all areas being very close to the "perfectly represented" colour. Here, the extreme under- and over-representation areas correspond to 5% fewer/more profiles in the training set than in the complete set.

Thick black line - The bias of the error for each network output

Blue lines - The standard deviation of the error for each network output

Crosses - The maximum and minimum error for each network output

Red lines - Instrument noise level

In each of these panels, the vertical axis denotes height and is in units of $z = -\log_{10}(\text{Pressure / hPa})$. The reason for this is discussed in section 4.3 of chapter 4. The scale runs from $z = -3$ to $z = 3$, or $p = 1000$ hPa to $p = 0.001$ hPa.

The final panel (bottom right) gives information about the training run of the network and some information about the network itself. The main graph shows the standard deviation of the network as a function of epoch. One epoch is measured as the time from one validation run to the next. The plotted value at each epoch represents the standard deviation of the network output that had the largest error during the validation phase. A number of details about the network are also given in this panel. These are:

η - The learning rate of the network

α - The momentum of the network

ν - The weight decay in the network (if non-zero)

best std. dev. - The standard deviation of the worst network output when the network couldn't be improved any further

Hidden Nodes - The number of hidden nodes in the network. If more than 1 layer of hidden nodes was present, this information is displayed as "n,m" where n is the number of nodes in the first hidden layer and m is the number of nodes in the next hidden layer

R1A.B1F.C1 - The current channel that is being modelled. Further information about the naming scheme can be found in section 2.3.1.

In order to be considered successful, it was decided that the network would have to have an error of less than the instrument noise. In order to do a good retrieval of atmospheric species, the forward model must be at least as good as the

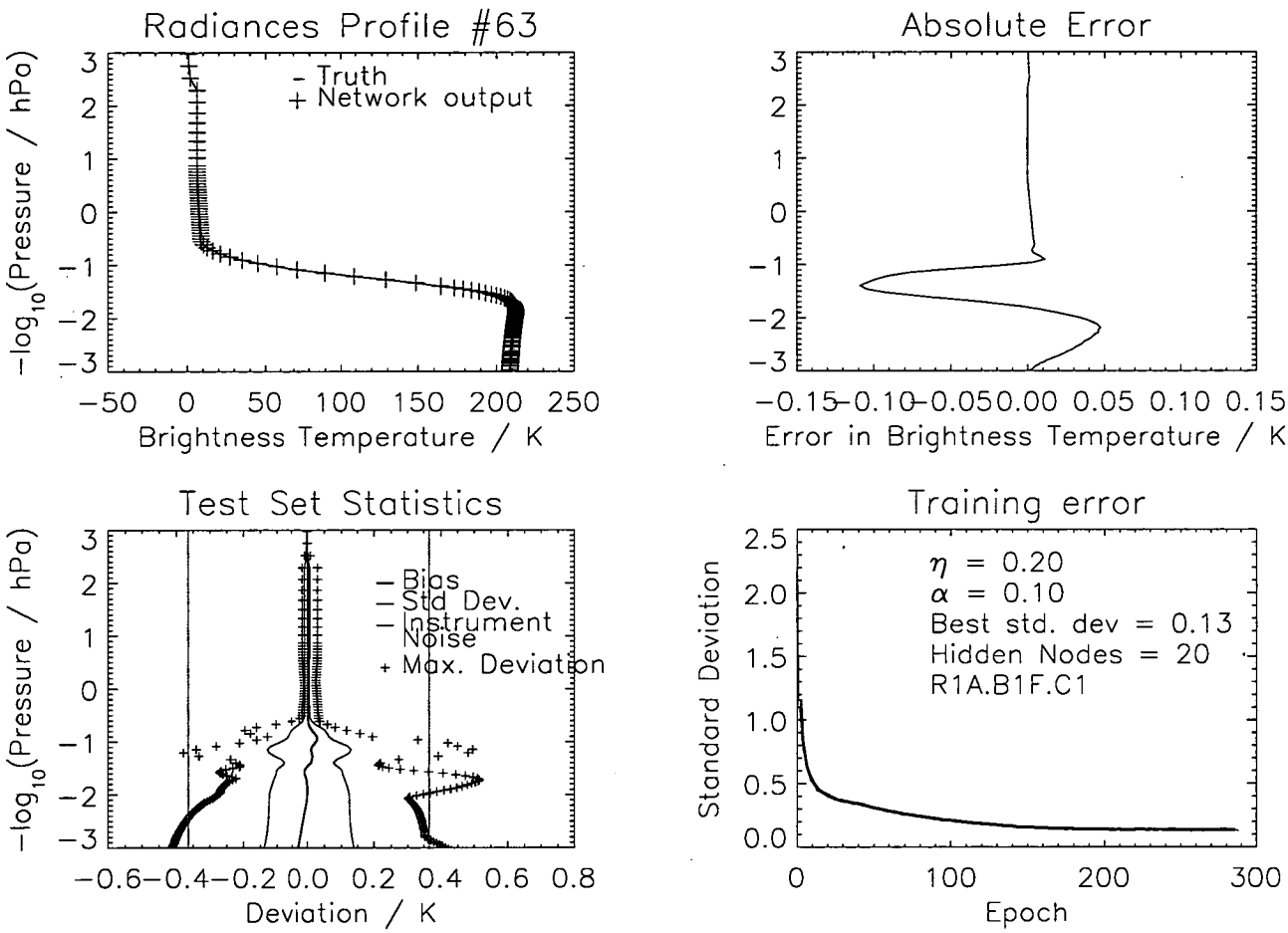

Figure 3.4: A sample results diagram. The diagram is split into four panels. The top left and right panel give a sample profile from within the testing set, the network output and the error on this output. The bottom left panel gives details about the overall test set, including the standard deviation, maximum and minimum errors and the instrument noise for the channel. The bottom right panel shows how the training of the network progressed and gives details about the setup of the network.

instrument noise levels and preferably, more accurate than half the instrument noise. For this reason, a network that, when trained, produces a standard deviation on its errors of less than the instrument noise is considered successful. If a network achieves a maximum standard deviation of less than half the instrument noise level it is considered to be well trained. Assuming the errors are Gaussian, insisting that the R.M.S. error is half the R.M.S. instrument noise implies that less than 3% of the profiles have errors exceeding the instrument noise for that tangent height.

Originally, the neural network was started using sigmoid transfer functions for all the hidden and output nodes. In order to ensure the best results, low values of η and α were used (0.2 and 0.1 respectively). The number of hidden nodes was increased or decreased in different runs, depending on how the network responded to different numbers. One of the first properly successful runs is given in figure 3.5. In this case, there are 55 hidden nodes in 1 hidden layer and the network returned a best error of $\sigma = 0.16$ K during testing. This shows that a neural network can act as a forward model in this case. It can be seen that the principal errors occur in the lower region of the profile. As the atmosphere is opaque to the instrument at these heights, these radiances won't generally be used in the observations vector of an assimilation model and as such the errors are less important than errors in the higher radiances.

Subsequent runs looked at the effect of varying several parameters in the network.

Varying the learning rate

Increasing the learning rate, η, allowed the network to rapidly reduce its error at the start of the training but in latter stages of training, the network tended to change the weights by too much, resulting in errors that didn't improve as rapidly over time and larger errors overall.

Varying the momentum

Increasing the momentum factor, α, means that more of the previous weight change is included in the current weight change. This should allow the network to overcome larger "bumps" in weight-space and allow a better final error. In practise, increasing the momentum rate too much again

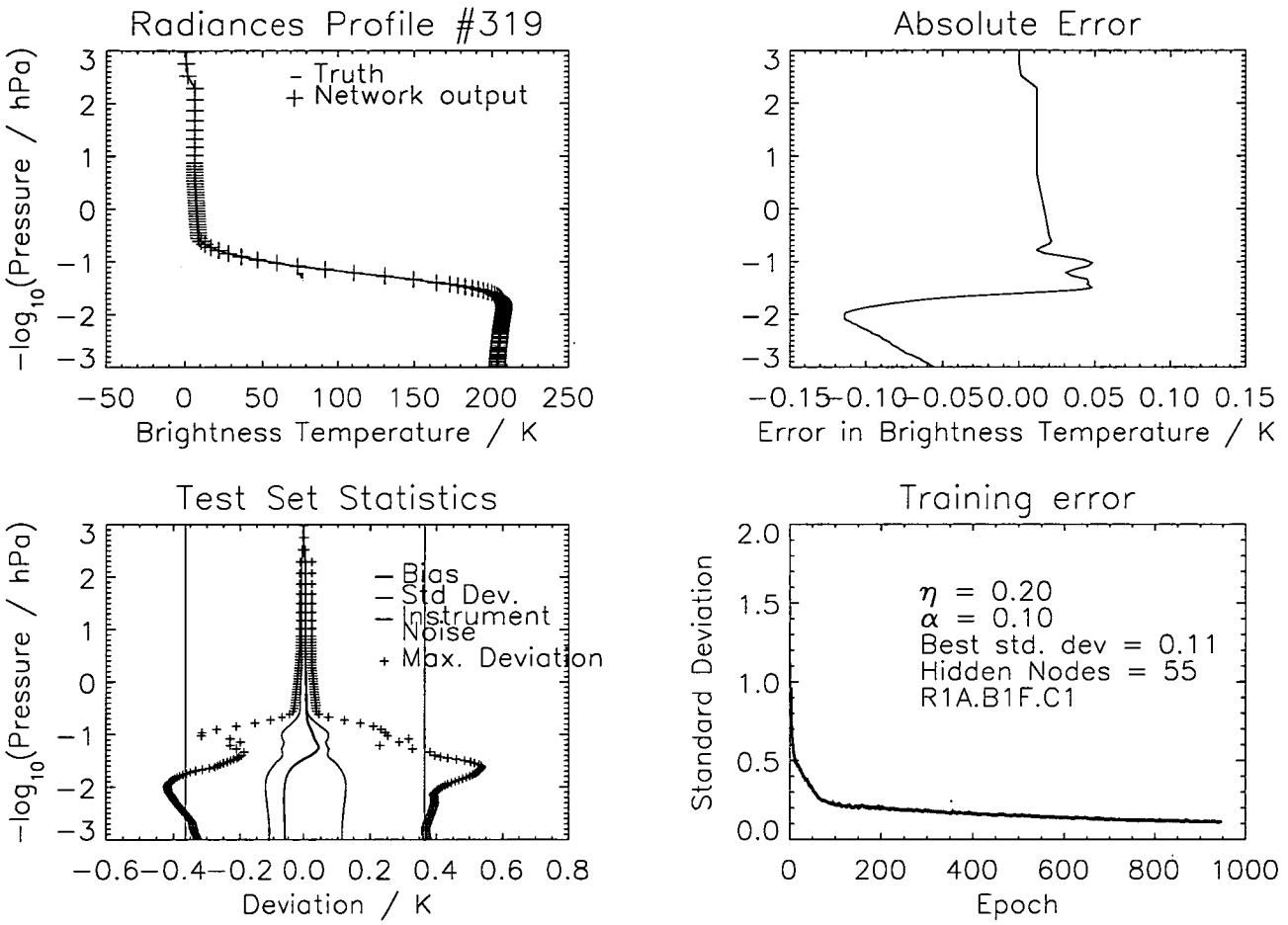

Figure 3.5: One of the initial successful runs (format is the described in section 3.5.1). This shows that a neural network is capable of learning to emulate the forward model in this (highly restricted) trial. In this channel, the instrument is sensitive between $z \approx -1.5$ and $z \approx -0.9$ ($\sim$ 30hPa to $\sim$ 10hPa). At heights greater than $z = -0.9$, the atmosphere is transparent at this frequency and the instrument measures background radiation. At heights below $z = -1.5$, the atmopshere is "blacked out" and the instrument detects radiation from (close to) $z = -1.5$.

results in too large changes in weight during the latter stages of training and larger errors overall.

Overall, it was found that learning rates of $\eta < 0.5$ and momentum rates of $\alpha < 0.3$ produced consistently good results in this case.

The final parameter that can be changed and should be looked at is the number of hidden nodes in the network.

3.5.2 The Effect of the Number of Hidden Nodes

Initially, the number of hidden nodes within the network was chosen heuristically - the number of hidden nodes for a particular run is chosen based on how the network reacted to the previous run. Systematically examining how the network responds to different number of hidden nodes allows us to check that the network is working consistently and that the optimum number of hidden nodes is chosen.

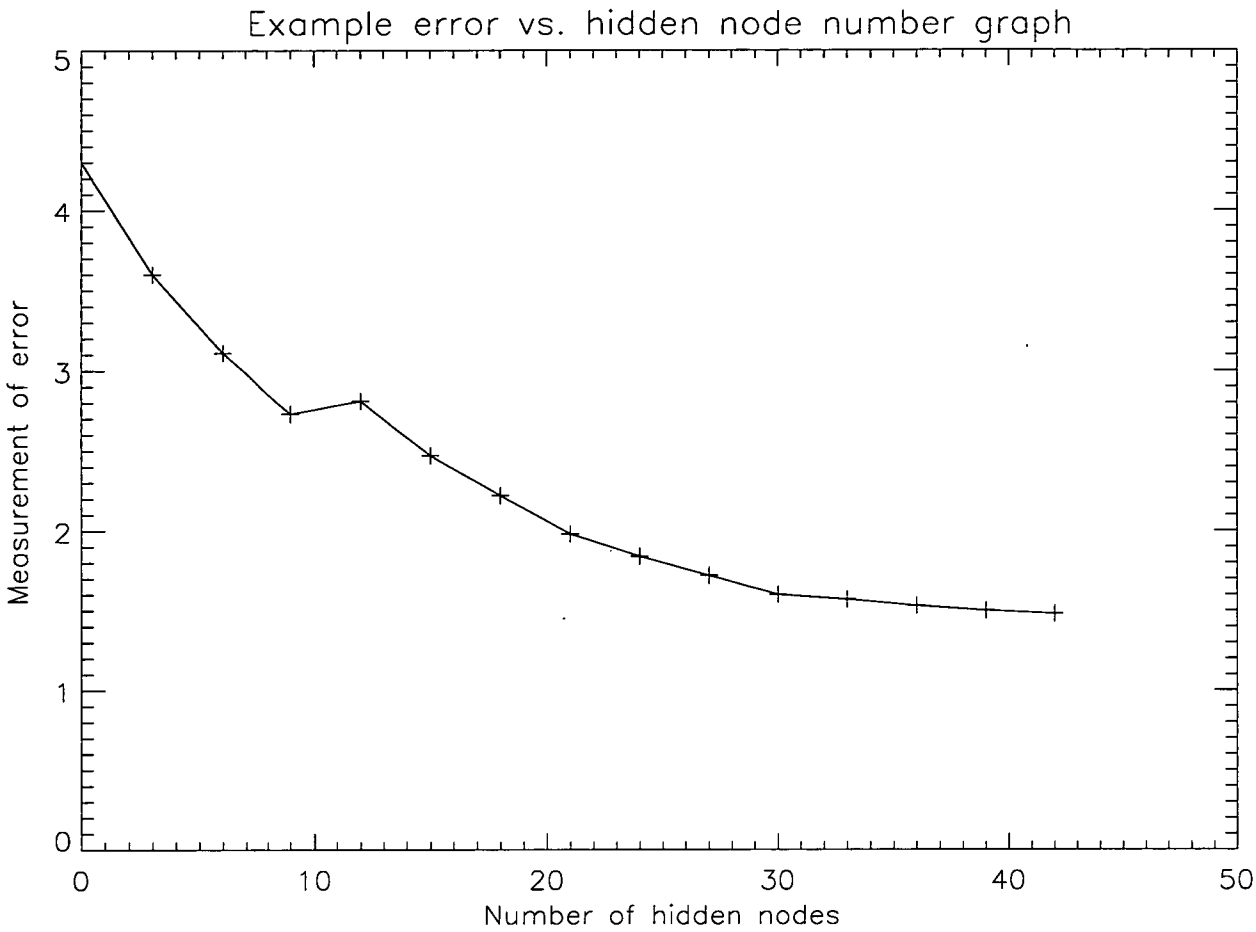

Figure 3.6: Example of an error vs. hidden node graph when the network is working properly. The ideal number of hidden nodes is a trade-off between speed and accuracy.

Consistently Working

Anthony and Bartlett (1999) suggest that there is an optimum number of hidden nodes for any particular network and that the error vs. hidden node number graph should be a decay graph, like figure 3.6. If a similar figure can be produced for this particular network, it would indicate that the network is indeed working properly. The optimum number of hidden nodes is then a trade-off between how accurate the network is required to be and the speed the network must be able to run at.

Optimum Number

> Using heuristic methods could result in missing a minimum on the error vs. hidden node number graph. If this were the case, there would be no real way of knowing that a promising region had been missed.

Investigating the effect of hidden node numbers is an easy task, if a little tedious. The basic principle involves starting the network with an arbitrary number of hidden nodes (typically zero) and letting it train. Once the training is complete, a test set is run through the network and the standard deviation is calculated for each network output. The worst of these standard deviations is recorded and the number of hidden nodes in the network is increased, the weights are randomised and the training starts again. This is repeated a number of times with varying numbers of hidden nodes. When this is complete, these standard deviations are plotted against node number.

Initially, the network was trained with no hidden nodes and each time the training was restarted, five more hidden nodes were added. This produces figure 3.7 where the bars on each point represent the maximum and minimum values across five runs. In this case, there is a sharp rise in the error from zero hidden nodes to 5 hidden nodes and then a steady decrease until around 20 hidden nodes. From 20 hidden nodes upward, the error remains approximately constant. This suggests that any additional hidden nodes above 20 are not being used in this case and so can be ignored. The sharp increase from zero to 5 hidden nodes is due to the 5 hidden nodes being unable to cope with the large number of inputs and outputs. In the case where there are no hidden nodes, all the inputs are directly connected to the outputs, and each output node can choose the most important inputs to it. However, in the case with 5 hidden nodes, these 5 hidden nodes must condense all the information from the inputs before passing it on to the outputs. The outputs in this case only have access to 5 bits of information, which is not enough to accurately calculate the radiances. The network therefore has a larger error than with zero hidden nodes. The small error bars also show that the network is well trained as the final network error is consistent across training runs.

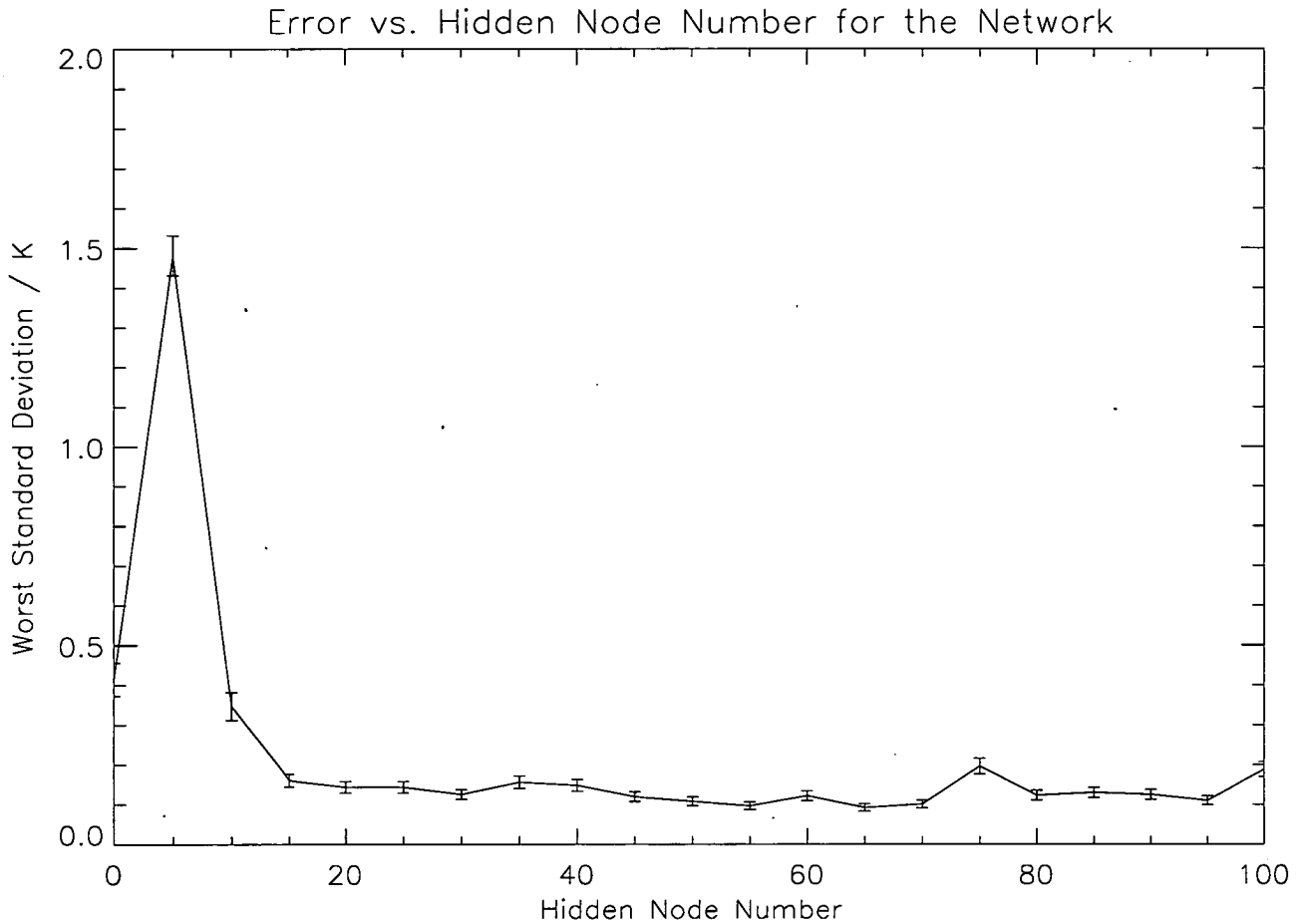

Figure 3.7: The error vs. hidden node graph for the forward model neural network. The error bars show the maximum and minimum across 5 neural network training runs. In this case, the graph is almost flat after around 20 hidden nodes. This suggests the best number of hidden nodes to have is around 20.

3.5.3 Improving the Initial Results

This section looks at several techniques that other people have employed to improve a network's ability or to reduce training times. There are many techniques that can be used to improve a neural network in some way, but only three will be considered here - using a weight decay parameter, using a non-linear error function and training the network using quickprop instead of backprop.

The first of these is to examine the use of a weight decay parameter. A weight decay parameter is an additional term to the weight update rule that subtracts a small percentage of all the weights in that layer from each weight, which means that smaller weights are favoured by the system. If large weights are present, the activation function can become nearly discontinuous - small changes in input will cause large changes in output (see e.g. Sarle et al. (1997) for more information about weight decay).

Adding a weight decay term to the neural network is technically easy, the difficulty coming with making the right choice. I simply used trial and error, guided by results from previous runs. One of the most successful tests is shown in figure 3.8. In this case, a weight decay coefficient of 0.01 was used. In comparison to figure 3.5 it can be seen that although the bias is improved when using weight decay, the network is not as well trained (seen by comparing the standard deviations of the test set in the third panel.). When no weight decay is used the standard deviations are better than when weight decay is used (0.16 K compared to 0.20 K in the case with weight decay). Examining the fourth panel may reveal a possible reason for this. In the weight decay case, the network is trained for a much shorter time (401 epochs compared to 950 epochs). Both networks are using the same criteria for stopping training (see section 3.4) which implies that in this case, training with weight decay is not as efficient as training without weight decay.

The second method of improving the network is using a better error function. It is suggested in Fahlman (1988) that using a nonlinear error function may increase learning speed of the network. The idea is that for small differences between true and network outputs, the error function should behave almost linearly but for bigger differences the error function should grow faster than linearly. This allows much greater learning at the start of training, when the errors are large, resulting in reduced training times. Fahlman (1988) used a hyperbolic arctangent

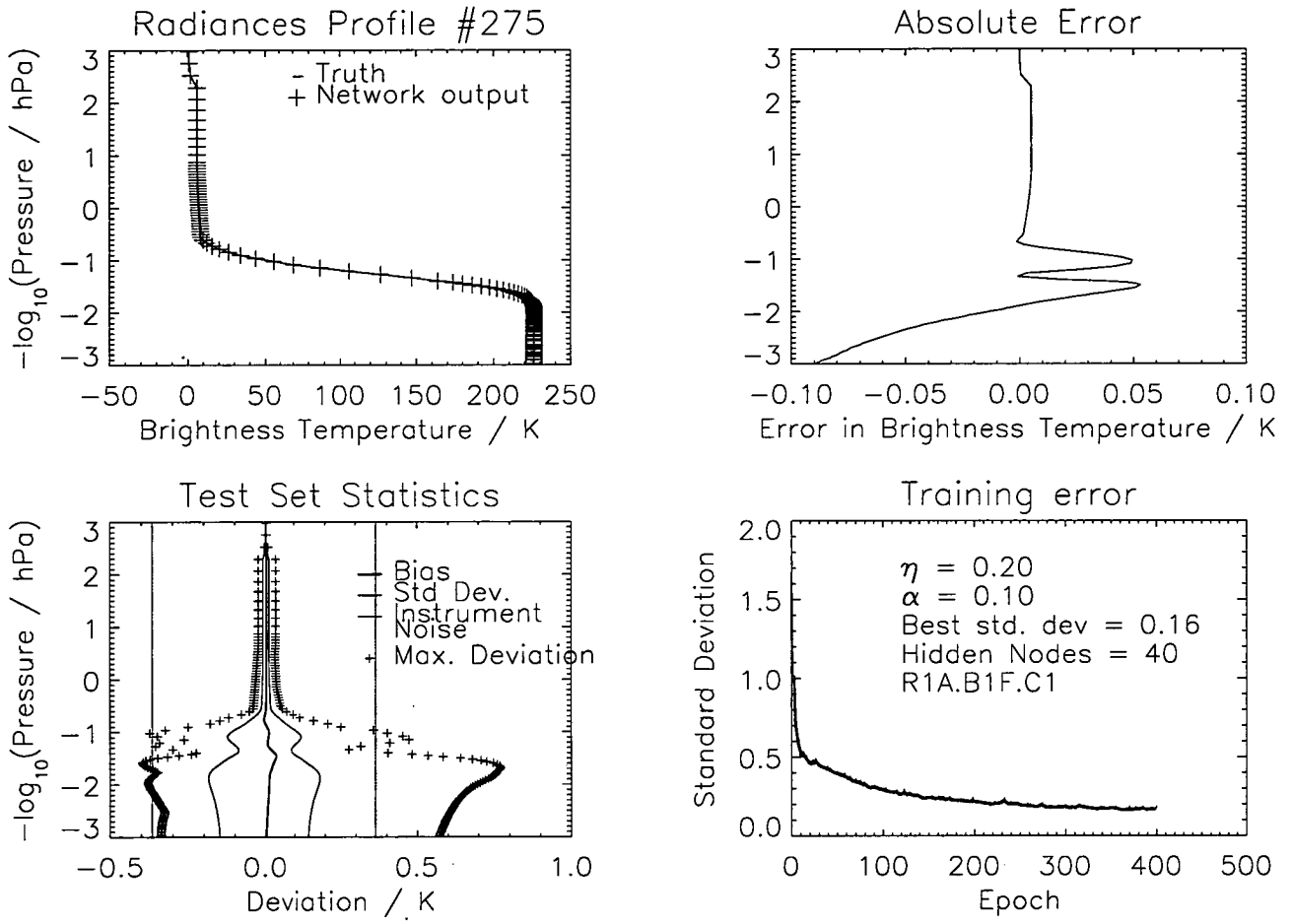

Figure 3.8: A neural network trained with weight decay included (format is the described in section 3.5.1). A weight decay coefficient of 0.01 is used in this case. This shows that for this network, using weight decay does not provide an advantage (when compared to figure 3.5).

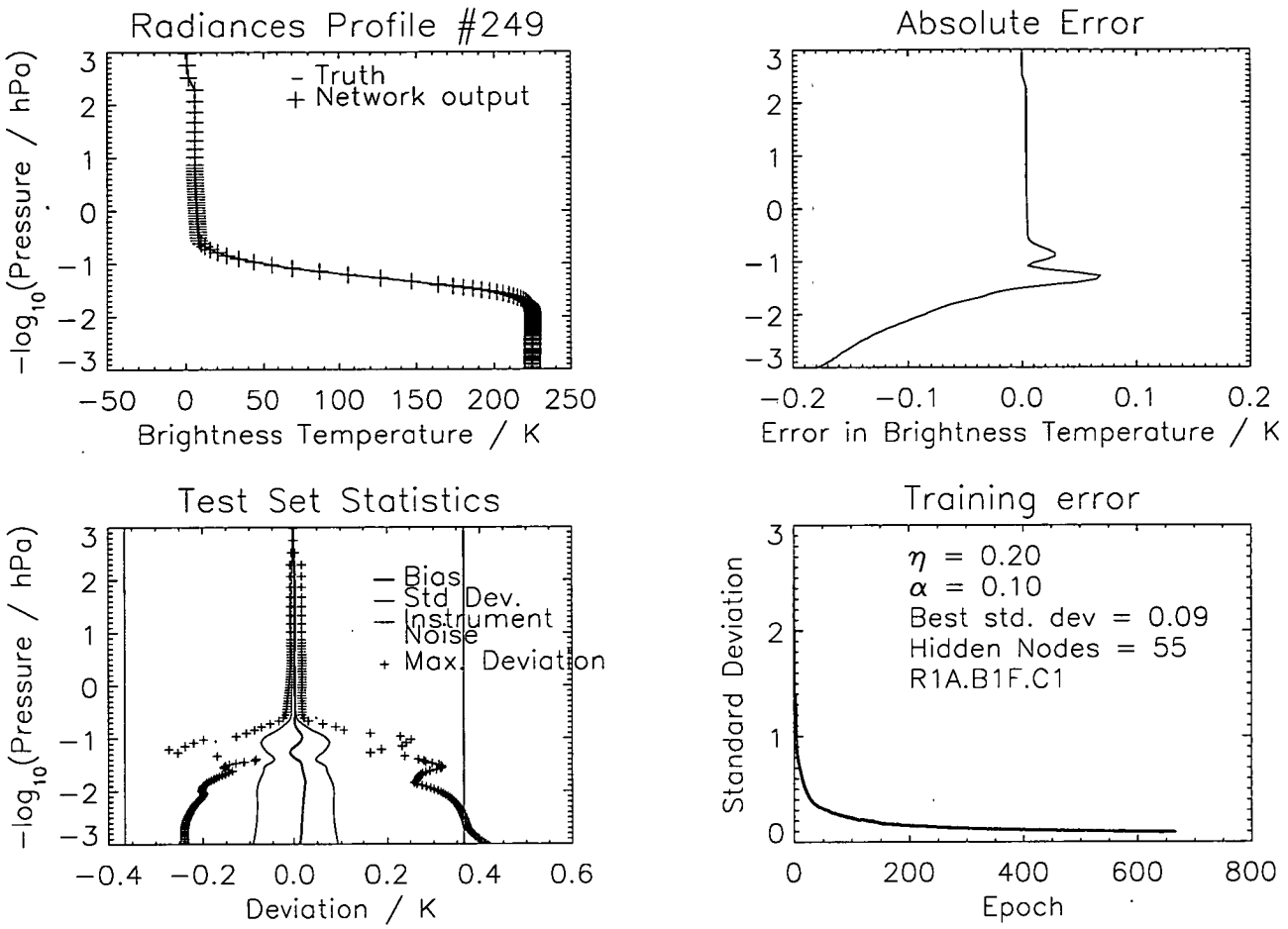

Figure 3.9: A neural network trained using a nonlinear error function (format is the described in section 3.5.1). Compared to using a linear error function (figure 3.5), the number of epochs needed to train the network is significantly reduced at no loss of precision for the network.

error function and achieved a 25% improvement in training times.

An example training run using a hyperbolic arctangent error function can be found in 3.9. Here, the important figure is the bottom right. This shows the training error as a function of epoch. When compared to 3.5, it can be seen that in this case, the training takes a shorter time (669 epochs compared to 950 previously) and the error has been slightly reduced (0.12 K compared to 0.16 K). Subsequent runs show that the training time is, in general, reduced when using a hyperbolic arctangent error function, with no loss of precision in the final trained network.

Another possible method of improving the training of the network is to train the network using quickprop instead of backprop - discussed in section 3.4. Quickprop should provide much quicker training and reduced error in the final network provided both assumptions, that weight space is parabolic and that the error vs. weight curve for each weight is independent of other weights, are met. In this case, using quickprop with very low learning rate resulted in the internal weights in the system rapidly diverging towards infinity. This suggests that one of the two assumptions is false. This is probably due to the assumption that the slope of the error vs. weight curve is independent for each weight. As previously stated, outputs close to each other in the network will depend on similar inputs in similar ways. This means that the errors for these outputs will be closely tied to the same weights from the input-to-hidden layers, invalidating this assumption. Once it was established that a well-trained network could not be produced using quickprop further tests weren't carried out.

3.6 Discussion

In this chapter we have looked at using a neural network as a forward model in a very limited set of cases. It has been shown that in these conditions, a neural network can be created that does function as a forward model. It has also been shown that using a nonlinear error function significantly improves training times of the network without affecting the precision.

Several things that are not included in this chapter, but have been investigated include changing the transfer function of the nodes within the network (section 3.3) and using different training algorithms. In the first case, several different transfer functions were tested, namely, the hyperbolic tangent transfer function and the linear transfer function. In the case of the hyperbolic tangent transfer function, the training was found to be very erratic, with the system weights often diverging to infinity, while the system was not as accurate when a finite result was produced. Using a linear transfer function the output nodes with a sigmoid function in the hidden nodes resulted in much higher errors during training while the training time was also increased.

Another possible way of improving the network is to use a different training scheme. Here, quickprop was tested but found to be unusable in this case. This is

thought to be due to the assumption, that the error vs. weight curve is independent for each weight, is false in this case. Other methods of training the network (e.g. using Bayesian learning techniques - MacKay (1995)) were not explored but could provide improvements to the results presented.

Some of the work carried out in this chapter can be applied to the full neural network. As quickprop proved unusable in this case, it was discarded as a possible training method for future neural networks. The results for learning rate (η) and momentum (α) can also be applied to future networks, keeping the value of both low. As the number of inputs is greater when tangent pressures are included, the results from the trials into hidden node number will not apply directly, but have proved the neural network is behaving as expected. The results from the tests into the number of hidden nodes also give a minimum bound to the required number of hidden nodes. If the neural network in this case requires at least 15 hidden nodes, it is a reasonable assumption that future, more complex networks will require at least this number of hidden nodes.

Other channels will be dealt with in chapter 5 but the work in this chapter should apply in other channels in band 1. Band 1 is centered on the 118.75 GHz oxygen line, which is much stronger than other spectral lines in the region and means the effect from other chemical species will be minimal. Other bands are centered on different spectral lines and will depend on the concentrations of other chemical species. For this reason, the work presented here should provide a basis for looking at these but will need substantially enhanced to deal with them.

Chapter 4

Tangent Pressures

4.1 Introduction

In the previous chapter, a neural network was developed that could act as a forward model for the EOS-MLS instrument in a very limited case. The temperature profile was the only input and it was assumed that a given radiance in a profile was measured at a constant pressure across all profiles. In reality, the pressure level that each radiance is measured at varies across profiles.

This chapter introduces the concept of tangent pressure levels and discusses why they are problematic for data assimilation schemes. Several methods of dealing with the problem are then explored and it is shown that a possible solution is to use a neural network retrieval.

4.2 What are Tangent Pressures?

For a limb sounder, the viewing geometry is shown in figure 4.1. The field of view can be scanned up and down, creating a series of measurements. Each measurement within a scan is called a minor frame and one complete set of measurements is a profile. On the ray at the centre of the field of view, the pressure at the point closest to the Earth is called the tangent pressure. For the EOS-MLS, one profile consists of 120 minor frames and the tangent pressures are spaced approximately uniformly from 1000 hPa to 0.001 hPa with a typical profile that looks like figure 4.2. Because of variations in the satellite attitude, the tangent pressures differ slightly but significantly from one profile to the next.

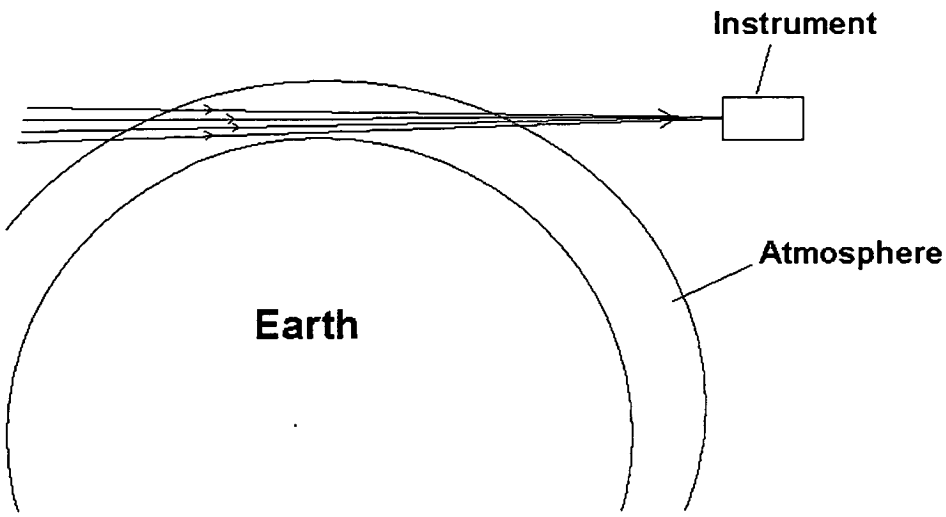

Figure 4.1: The basic viewing geometry of a limb sounder.

Tangent pressure levels are related to the geometric heights by way of the hydrostatic equation (see section 4.3) and hence can be calculated within error limits from the satellite position and scanning angle, provided the temperature and water vapour profiles are known. In most cases, these profiles are not known and so the tangent pressures must be deduced in some other way.

The value of each tangent pressure level is required in any realistic forward model as all retrievals are done in (log) pressure space. Normally, tangent pressures are part of the retrieval process, as the instrument returns data in geometric space, but the retrieval is done in (log) pressure space, thus it is necessary to retrieve tangent pressures to make the retrieval process optimal. At the end of the process, tangent pressures are not normally reported as they serve no purpose for the scientific end-user.

In section 4.4.1, it is shown that treating tangent pressures as fixed does not give adequate accuracy. To overcome this, either the network must be supplied with enough ancillary data to allow acceptable radiances to emerge or the tangent pressures must be retrieved outside the assimilation scheme. These possibilities

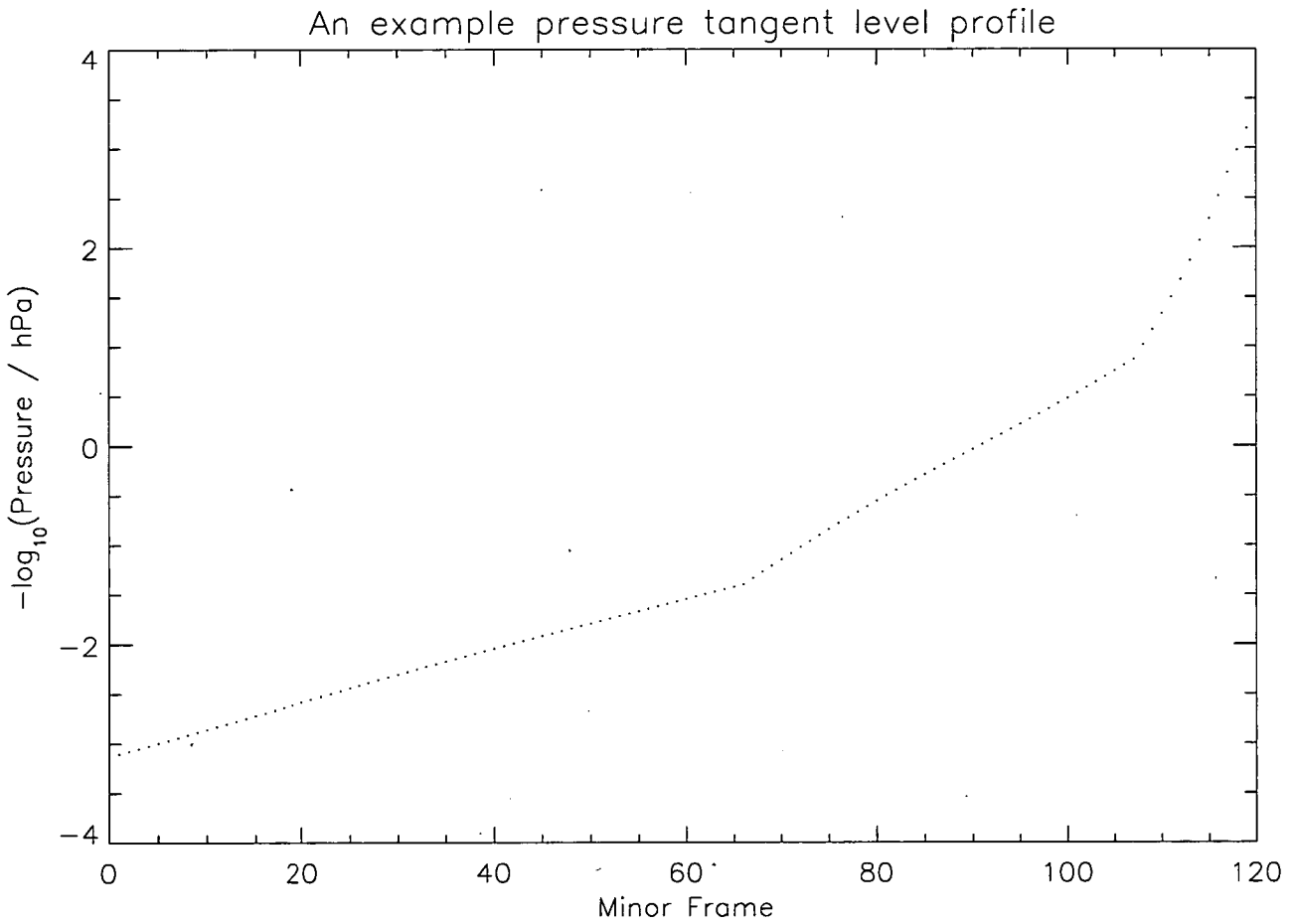

Figure 4.2: A sample tangent pressure level profile for the EOS-MLS. Each dot represents one minor frame. There are 120 minor frames per profile.

are dealt with in sections 4.4.2 and 4.5 respectively.

4.3 The hydrostatic equation

This section is derived from the document by Pumphrey (1999). The hydrostatic equation is given by equation 4.1 where p is the pressure, ρ is the density and g is the acceleration due to gravity and h is the geometric height.

$$\frac{dp}{dh} = -\rho g \tag{4.1}$$

For an ideal gas, $p = \rho R_u T/M$. Here, R_u is the universal gas constant, T is the temperature and M is the mole mass of the gas. Substituting this into equation 4.1 leads to equation 4.2.

From equation 4.2, assuming that fractional changes in T are small, we see that pressure is approximately an exponential function of height. As this relationship would be exact in the case where g and T are constant in height, a convenient vertical coordinate is given by $z = -\log_{10}(P/1\text{ hPa})$ (as used previously in section 3.5.1 of chapter 3). This can be defined in terms of the geometric height using equation 4.3.

$$\frac{dp}{dh} = -\frac{Mg}{R_u T}p(h) \tag{4.2}$$

$$\frac{dz}{dH} = \frac{g_0}{RT\ln(10)} \tag{4.3}$$

Here the geopotential height, H, is used as opposed to the geometric height, h, and g_0 is an arbitrary constant designed to make $H \approx h$ for altitudes close to zero. The geometric height, h, can be converted to the geopotential height, H, using standard formulae found in Wright Jr. (1997). Assuming that M is constant for the height of interest, $R = R_u/M$ can be considered constant.

When doing a retrieval, it is assumed that T varies linearly with z between adjacent pressure levels, z_0 and z_1, this can be written as $T(z) = T_0 + B(z - z_0)$, where $B = (T_1 - T_0)/(z_1 - z_0)$. Integrating equation 4.3 with this relationship leads to equation 4.4, which can be rearranged to 4.5, where $T_m = (T_0 + T(z))/2$.

$$H - H_0 = \frac{R\ln(10)}{g_0} \int_{z_0}^{z} \{T_0 + B(z' - z_0)\} dz' \tag{4.4}$$

$$z - z_0 = \frac{g_0}{R\ln(10)T_m}(H - H_0) \tag{4.5}$$

Using this formula, it is possible to find all values of the tangent pressures, z, provided the geopotential heights, the temperature profile and an initial z_0 is known. Unfortunately, the temperature profile and z_0 are not known in radiance assimilation and are part of the retrieval hence a different method of getting z's must be found.

4.4 Possible Solutions to the Tangent Pressure Problem

This section looks at possible ways of avoiding using tangent pressure information in a neural-network-based forward model. This may be possible due to the "black box" nature of the neural network. Here, two possible methods of doing this are examined: assuming invariant tangent pressures and using geometric height information instead of tangent pressures as inputs for the neural network.

4.4.1 Invariant Tangent Pressures

The first, and simplest, solution to the problem is to assume tangent pressures are fixed across profiles. If this were the case, the only inputs that would be required by the neural-network-based forward model would be the temperature profile[1], much like the model in chapter 3. A necessary (but not sufficient) condition for this to be considered a good approximation is that the spread of the tangent pressures at one level must be relatively small. This would mean the variance in the tangent pressures would not have a drastic impact on the final radiance. A relevant measure of invariance is whether the spread of one tangent pressure level across profiles is much less than the difference between consecutive tangent pressure levels within one profile.

[1] and any other species to be included in the forward model calculation

Figure 4.3 shows the spread for several typical levels from within the training set. As can be seen, the spread is much greater than the difference between the levels. This shows that treating tangent pressures as constant across profiles will result in some radiances being attributed to the wrong minor frame tangent pressure, implying the variation in tangent pressures must not be ignored. To ensure that this is the case, several small trials were carried out (not presented here). The results show a network error in the region of $\sigma = 1.2$ K, which is about four times the instrument error and hence unacceptable. Thus tangent pressures cannot be treated as fixed.

4.4.2 Using Geometric Height Information

A second possibility is to use geometric height information in the neural network forward model. Geometric heights are available as part of the level 1 ancillary data and are deduced from the instrument's viewing geometry. It may be possible to use geometric heights in a neural network forward model as tangent pressures are linked to them (see section 4.3). In essence, this is adding an additional step (converting geometric heights into tangent pressures) into the forward model process which would happen implicitly within the neural network.

Figure 4.4 shows the network output of a sample run using geometric heights instead of tangent pressures and takes the same form as previously described[2]. Where geometric height information is used, the error is around 0.75K, about twice the expected instrument noise level for this channel and hence still unacceptable.

4.5 Acquiring Tangent Pressure Information

In the previous section, it was shown that tangent pressure information must be found for a neural network forward model. In this section, several methods for acquiring the tangent pressures are presented. First, traditional methods are shown to be insufficient and then retrieving tangent pressures using a neural network is explored.

[2]See section 3.5.1

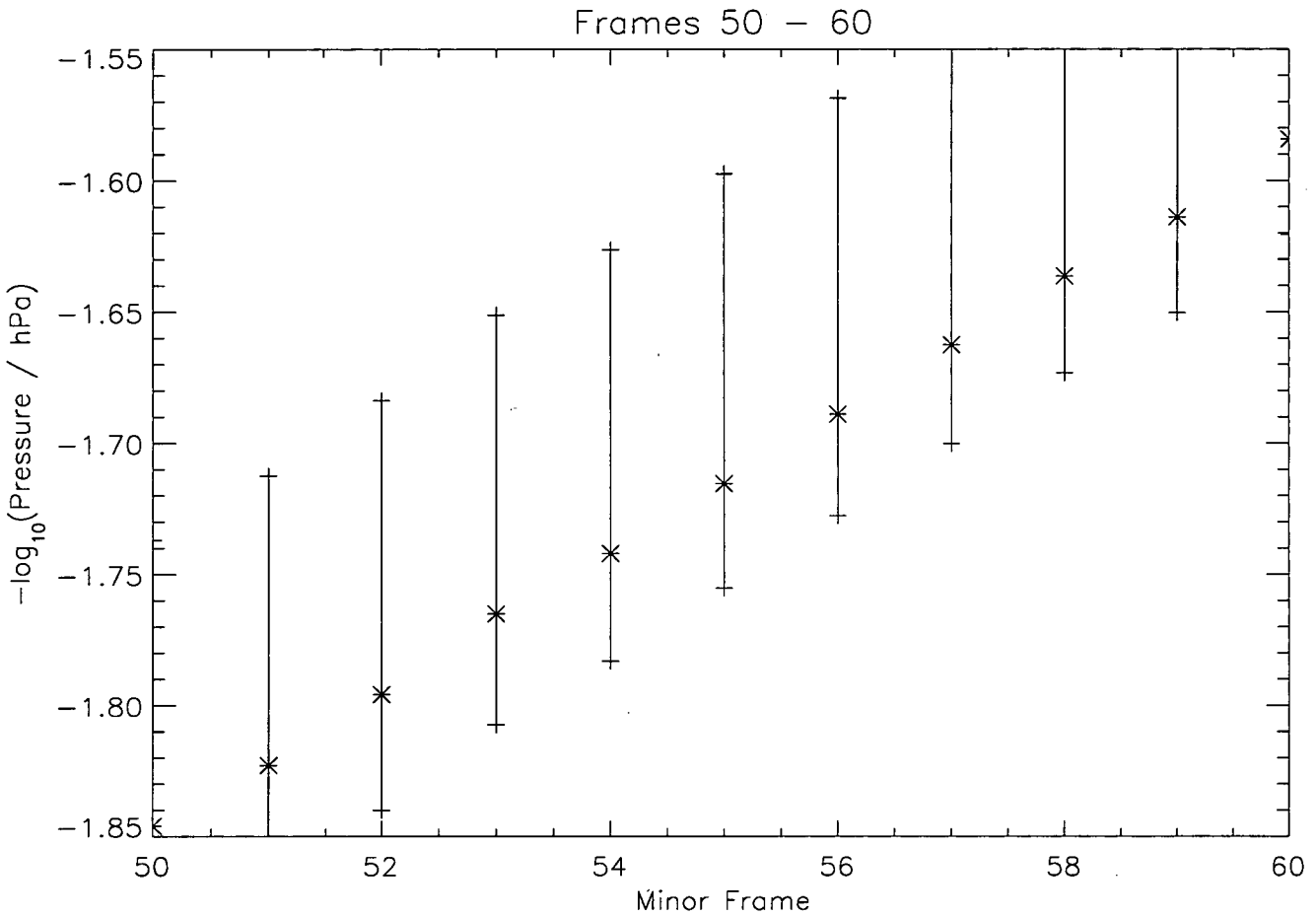

Figure 4.3: Tangent pressure variation. The stars give the mean tangent pressure value across 1500 profiles for minor frames 50 to 60. The vertical lines show the extremes for that minor frame over the same series. This shows that the variation in tangent pressure value within one minor frame is much greater than the difference between adjacent minor frames

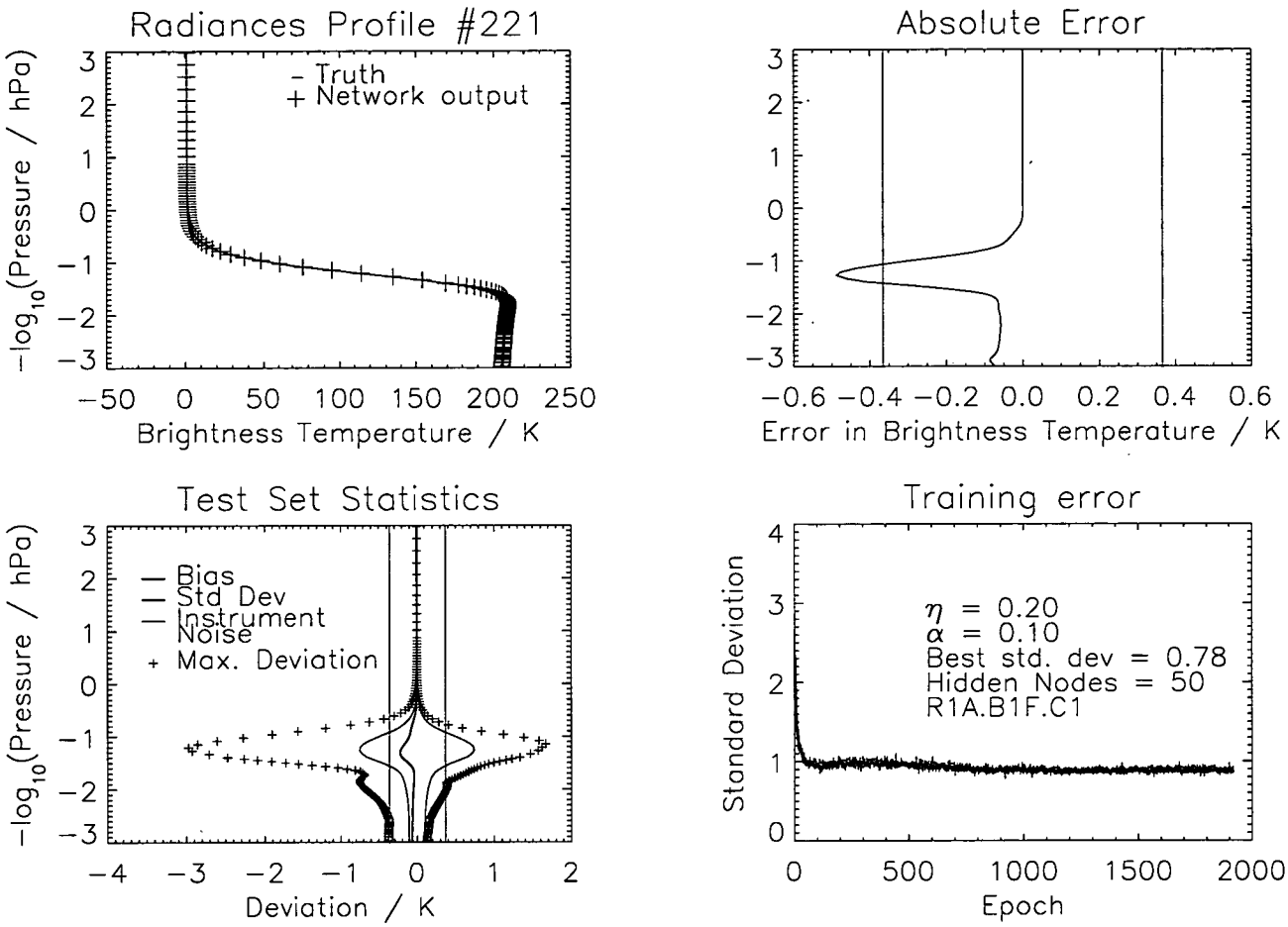

Figure 4.4: A sample neural network training run using geometric heights as inputs as opposed to tangent pressures (format is the described in section 3.5.1). The format of the figure is the described in section 3.5.1 of chapter 3. The error in the test set is 0.75 K which is approximately 3 time the instrument noise in this channel.

4.5.1 Traditional Retrieval

The first possibility for retrieving tangent pressures is to do some form of traditional retrieval. This could be done in one of several ways. One way is to wait until the level 2 products become available, allowing the "official" tangent pressures, and as such the best estimate, to be used. This additional accuracy comes at a cost of timeliness. It can take much longer for level 2 products to become available than level 1 products.

The cleanest option, if it is feasible, is to incorporate the tangent pressures into the state vector of the general circulation model (GCM) that is being assimilated into. This involves supplying the measured tangent heights and geometric heights and using the model's profile of temperature and geopotential height (GPH) to determine the tangent pressures. These tangent pressures are then used within the state vector that is to be updated during the assimilation process. This way, the estimate of the tangent pressures gets better while the assimilation step progresses. Once the vector is updated, the tangent pressures can be discarded.

This method should be relatively simple to achieve as the assimilation model does contain temperature, water vapour and reference GPH but there are technical difficulties in adding tangent pressures to the state vector of the assimilation model (Feng (2004)).

The final possibility is to perform a mini-retrieval outside the model. There are several problems associated with this. The major problem is that this introduces additional a-priori information into the data assimilation scheme thus violating the major reason for using radiance assimilation in the first place. Along with this, there are several other reasons why a traditional retrieval is not good in this case. One of these is the speed element. A traditional retrieval is done iteratively and may require a number of iterations before the result is optimal. Each iteration is itself a complicated process which is very computer-intensive. This will hold up the assimilation process as this must be done before the assimilation can be started.

4.5.2 Neural Network Retrieval

As traditional methods have been shown to be inadequate, a different approach must be found. Jiménez (2003) showed that it is possible to retrieve species

profiles from a limb sounder using neural networks and so it is expected that a neural network retrieval should be possible in this case. Here, several tests will be presented that show it is possible to retrieve tangent pressures using a neural network.

In order to be successful, the retrieved tangent pressures should have comparable errors to results results produced using conventional optimal estimation methods. These errors in a standard retrieval have a standard deviation of $\sigma < 50$ m for most minor frames within a profile (derived from Filipiak (1999)). The criteria for success using a neural network was that a standard deviation of $\sigma = 50$ m. Using equation 4.6, where Δh is the error in (geometrical) height, Δz is the error in (log) pressure coordinates and s is a scale height, an error of $\sigma \approx 0.003$ in tangent pressure in $\log_{10}$(Pressure / hPa) units is acceptable, assuming a scale height of 7.5km.

$$\Delta h = \Delta z * s * \ln(10) \tag{4.6}$$

The network used is, as before, a simple multi-layered perceptron trained using back-propagation. The outputs are the minor frame tangent pressures within the profile, and the inputs are the radiances from one or more channels.

The training data are a set of radiances generated from the same temperature data used in chapter 3, but the tangent pressures are allowed to vary across profiles here. As previously discussed, the training data has been shown to have a good distribution across the expected input-output space. The tangent pressures are randomly distributed around expected values and also give a good distribution across the expected range. Radiances were generated for all channels across band 1.

An initial training run is shown in figure 4.5. Here, the minor frame number is plotted on the vertical axis. For each case in the test set, the difference at the end of training between the retrieved tangent pressure and the true tangent pressure is calculated and plotted as a dot on the graph. The red line shows the standard deviation of the outputs at each height.

As can be seen by this plot, the most accurately retrieved region is in the middle of the profile (network outputs 40 to 70, around $z = -2$ to $z = -1.2$ or 100 hPa to 15hPa, where the channel used in this retrieval is most sensitive) with a standard deviation around 0.002. Using equation 4.6, this is equivalent to

an error of around 35 m. This shows that, in this region, a retrieval of tangent pressures within error is possible. Further regions can be improved by changing the inputs of the network to use a reduced profile from several channels within band 1 and band 32. Band 32 is a wide band with 4 channels centered around the same oxygen line as band 1 (figure 2.5 in chapter 2). This allows measurements much deeper in the atmosphere than using band 1 alone. This results in the network having 200 inputs made up of minor frames from different channels in band 1 and band 32. The minor frame numbers from each channel used can be found in table 4.1.

Band	*Channel*	*Minor Frames Used*
B1	1	40:60
	3	50:70
	6	55:75
	8	65:80
	10	75:85
	11	80:90
	12	80:100
	13	95:120
B32	1	0:30
	2	20:50

Table 4.1: The scan points used from different channels to construct the reduced profile.

Figure 4.6 shows the outcome of a training run using this reduced profile. Here, the results are again encouraging. The error across the entire profile has been reduced, especially in the extremities. As expected, the largest errors are still near the bottom of the profile because all the channels of the instrument are saturated at this height and so give very little information. At the bottom of the profile, the standard deviation is now around 0.015 ($\sim$ 260 m) but in general the standard deviation is less than 0.002 ($\sim$ 35 m). This shows that it is possible to retrieve tangent pressures using a neural network.

Using Geometric Heights

Geometric heights are derived from instrument pointing information and are related to the tangent pressures. In an optimal-estimation retrieval, they are used

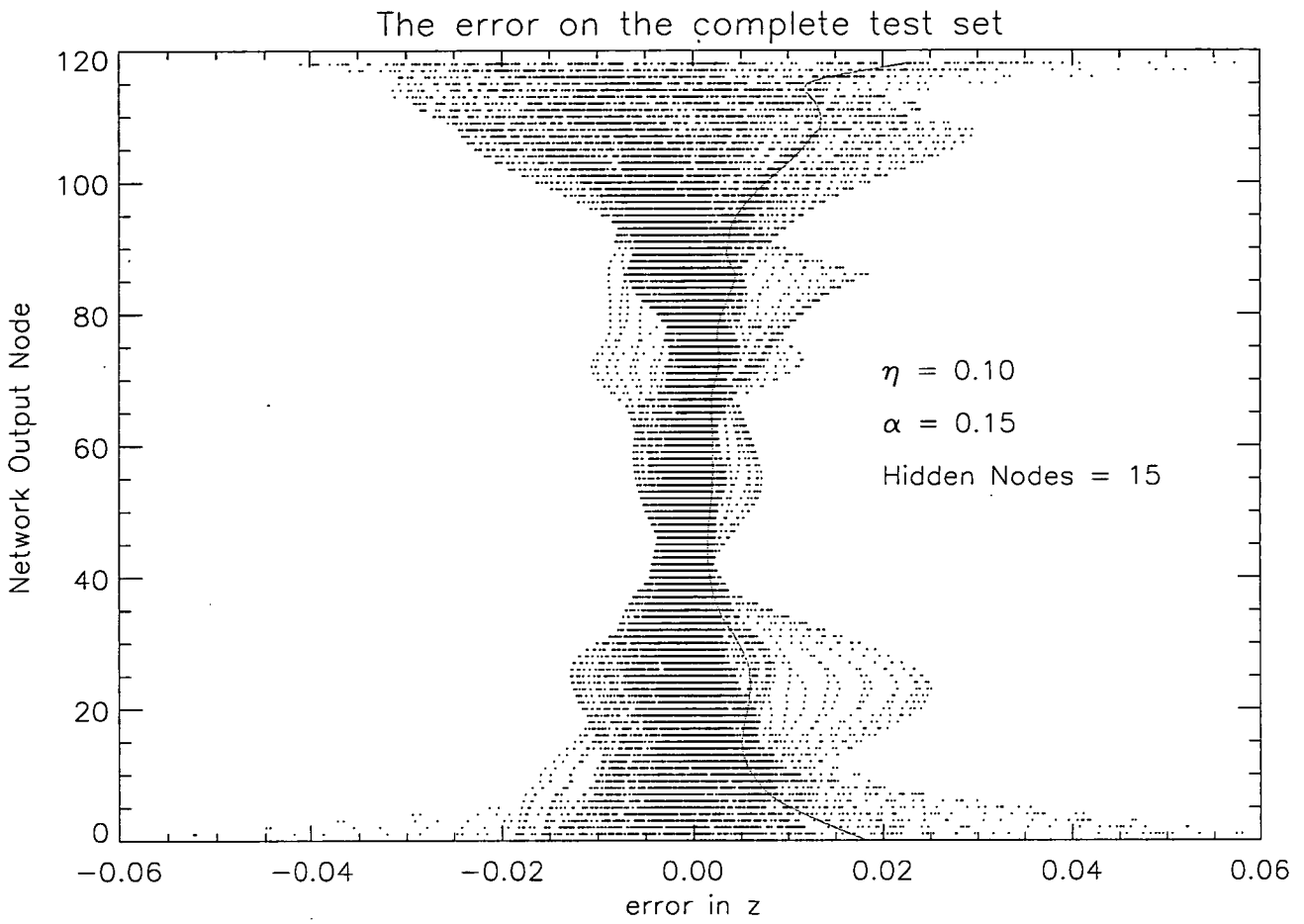

Figure 4.5: An attempt to retrieve tangent pressures using a neural network. This shows the error across the entire test set across all network outputs. The best results come around network outputs 40 to 60 which correspond to the knee of the radiance profile, where the instrument is gathering information. The red line shows the standard deviation for each network output. The large errors at the top of the profile are due to the instrument receiving very little radiation at this height, while the large errors at the bottom of the profile are due to the instrument being saturated, thus containing very little information about lower heights.

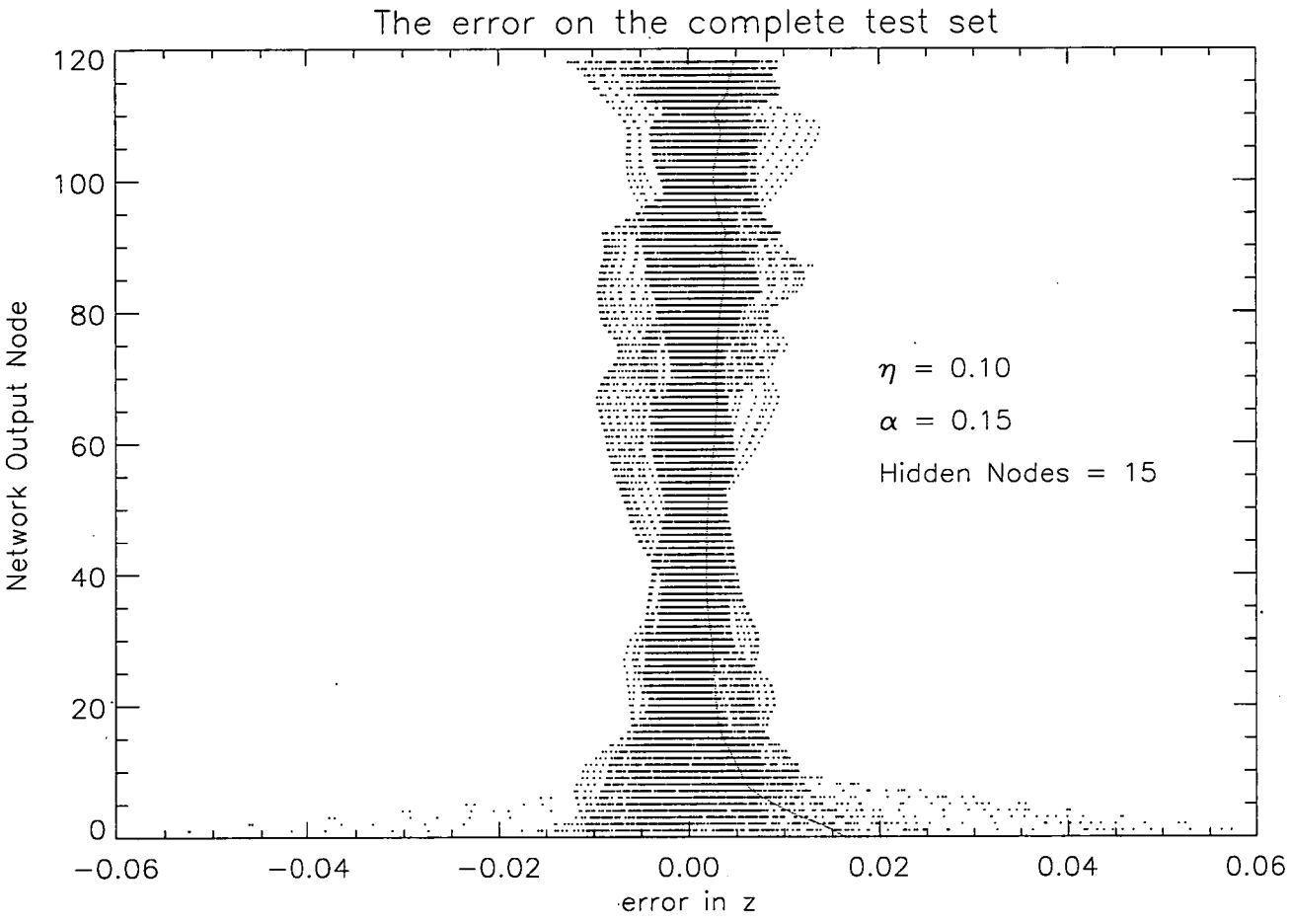

Figure 4.6: The error on retrieved tangent pressures using a neural network with a reduced profile created with Band 1 and Band 32 radiances. The red line shows the standard deviation for each network output.

as they improve the accuracy of the retrieved tangent pressures (Livesey and Wu (1999)). It is possible they may improve the accuracy of the neural network retrieved tangent pressures.

To test this, several neural networks were trained with geometric heights as inputs. These networks consisted of 320 inputs (200 radiance inputs as before and 120 geometric heights) and 120 outputs. It was found that the error across all the outputs was very similar to the case where no geometric heights were used. As the neural network without geometric heights produced results within expected the error range, the idea of using geometric heights in the neural-network retrieval was not followed up further.

4.6 Dealing with Noisy Radiances

Until now, this chapter has been dealing with retrieving tangent pressures from clean (i.e. error-free) radiances. In reality, different channels / bands have different noise levels associated with them. In this section, using noisy radiances within a neural network retrieval of tangent pressures is considered.

Neural networks are, in general, quite good at handling noisy inputs (e.g. Braspenning et al. (1995)). There are two possible routes for dealing with noise, both of which will be dealt with in this section. One is to train a network using clean inputs and then use the noisy inputs during the testing phase (with normally distributed random noise assigned to each input value in the test set), as was done in the previous section when retrieved tangent pressures were used in a forward model. The other way is to train the network using noisy inputs.

Both of the ways mentioned have advantages and disadvantages: training the network using clean inputs has already been done and so requires no additional work, just feed the noisy inputs in and work with the outputs. The disadvantage of this procedure is that the network may produce less accurate results than training with noise included in the inputs due to unexpected noise characteristics.

Training a network using noisy inputs is more time-consuming as noise must be generated for each input, each time the network is trained on a profile. This process slows down the training cycle substantially, but may result in better retrievals in operation.

Each of the methods described have been looked at and the results are pre-

sented here.

4.6.1 Training Using Clean Radiances

In this section, the idea of using clean radiances to train a network and then retrieving using noisy radiances is examined.

As several networks have already been trained using clean radiances (see section 4.5.2), it is a trivial task to use one of these networks to evaluate the effect of noise. Tests were run using a neural network with 200 inputs, 15 hidden nodes in one hidden layer and 120 outputs as this proved the most successful configuration when no noise was added to the inputs.

For the purposes of this experiment, 1000 profiles were used. These had never been seen by the network before, and represent a good cross-section of expected profiles. First, clean radiances were used to retrieve tangent pressure levels to use as a base for comparisons. The results of this can be found in figure 4.7, which shows the error on each tangent pressure (network output) for each retrieved profile. This is similar to figure 4.6. The network was then run with several levels of noise. The results are summarised in table 4.2. Here, the error column gives the RMS error across all network outputs across all profiles. As can be seen in figures 4.7 - 4.10, the majority of these errors are due to the minor frames at the lower- and upper-most minor frames, while the central portion of the profile has much better errors, as expected. This shows that a network trained with clean radiances can retrieve tangent pressures well when the inputs have low enough noise.

Run	Noise Used / K	Error	Height Error / m	Figure
A1	0.0	0.004	69	4.7
B1	0.4	0.005	86	4.8
C1	1.0	0.006	103	4.9
D1	5.0	0.012	207	4.10

Table 4.2: Training runs carried out using a network trained with clean radiances. The error column is the RMS error across the complete set.

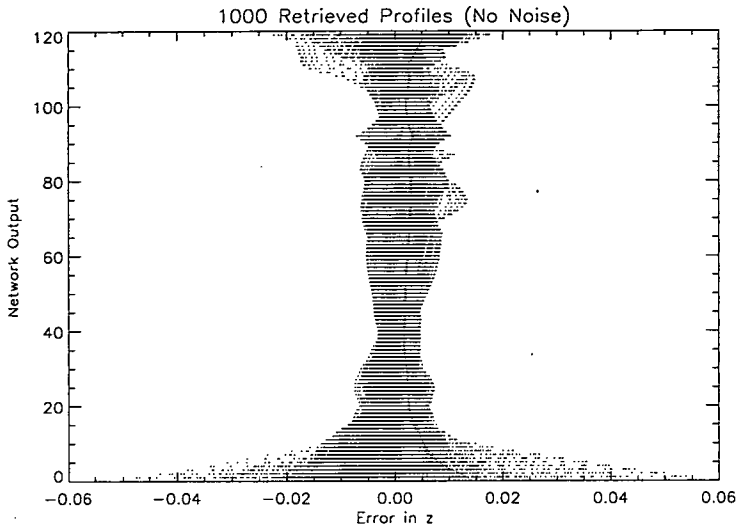

Figure 4.7: Retrieved tangent pressure levels for run A1.

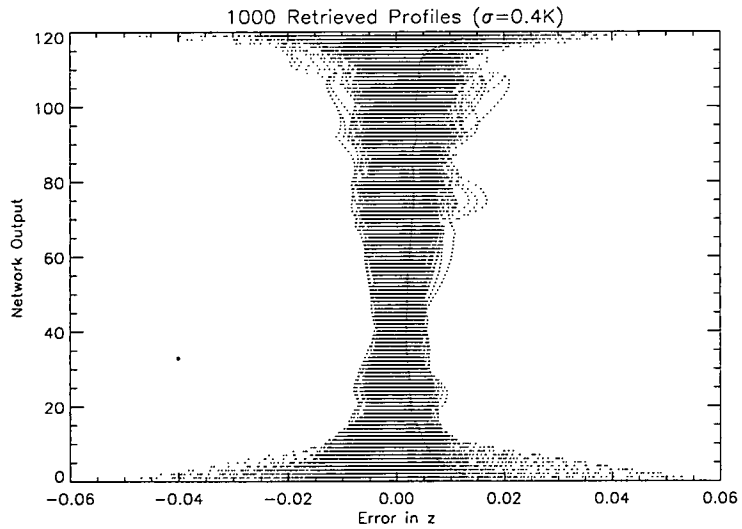

Figure 4.8: Retrieved tangent pressure levels for run B1.

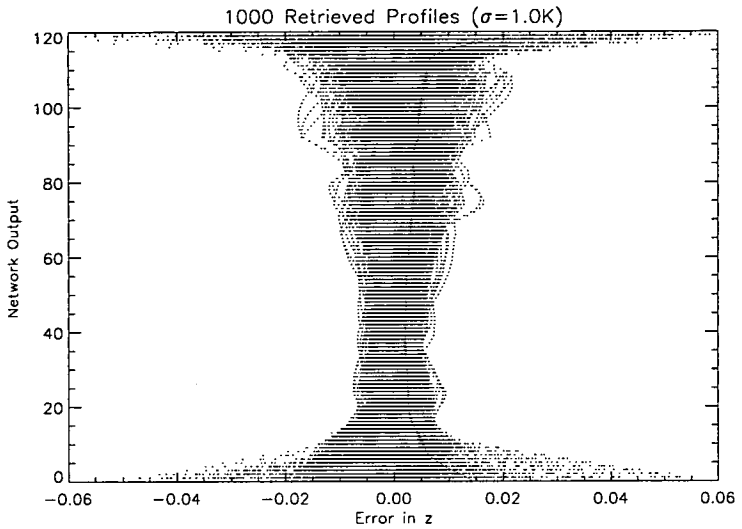

Figure 4.9: Retrieved tangent pressure levels for run C1.

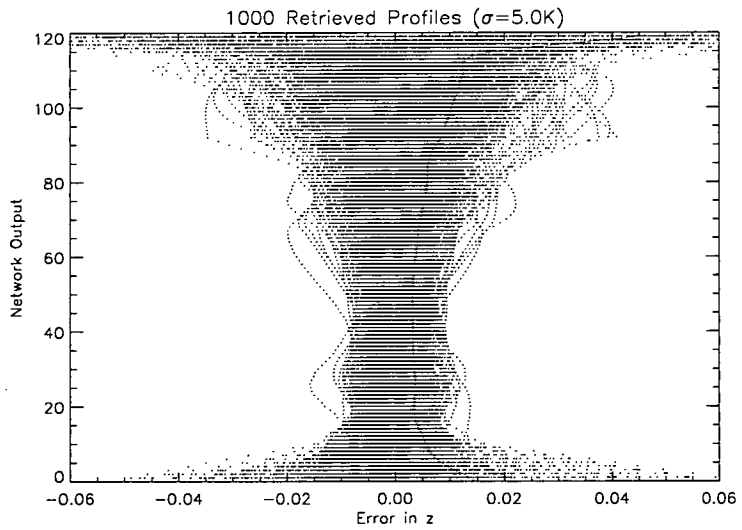

Figure 4.10: Retrieved tangent pressure levels for run D1.

4.6.2 Training With Noisy Radiances

This section looks at how a network can be trained using noisy radiances and how that affects the retrieval. The architecture of the network is the same - 200 inputs in the form of a reduced profile as described in section 4.5.2, and 120 outputs for the tangent pressure levels. The number of hidden nodes was chosen to be 15 in one hidden layer, as this worked well when dealing with clean radiances.

Each time a training profile is read in, a normally distributed noise, with a standard deviation at the noise level being examined, must be generated for each individual radiance. This can take a long time when each training epoch runs for 7500 profiles, and there can be hundreds of epochs. For this reason, training with noisy radiances can take much longer than training with clean radiances and so fewer tests were carried out in this case.

As before, several noise levels were investigated and these are summarised in table 4.3. As can be seen, when the network noise is sufficiently low (up to 1.0K), there is no advantage to training the network with noisy radiances. At large noise levels however, the network performs significantly better when trained with noisy radiances.

Run	Noise Used / K	Error	Height Error / m	Figure
A2	0.4	0.005	86	4.11
B2	1.0	0.005	86	4.12
C2	5.0	0.007	121	4.13

Table 4.3: Training runs carried out using a network trained with noisy radiances. The error column give the RMS error across the complete set.

This section has discussed dealing with noisy radiances in two different ways. The first method involves training a network using clean radiances and then subjecting that to radiances with noise and seeing how it copes. The second method involves using noisy radiances during training. It was shown that when the noise on the radiances is small (standard deviation of less than $\sigma = 1.0$ K), both approaches work equally well. When the noise grows to significant levels ($\sigma = 5.0$ K), training a network with noisy radiances becomes worthwhile.

As has been stated, training a network with noisy radiances becomes more expensive as many thousands of noise levels must be generated. The expected

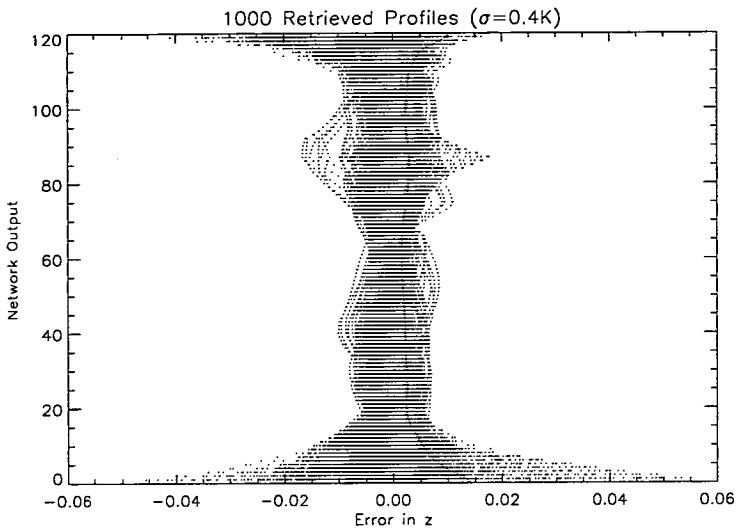

Figure 4.11: Retrieved tangent pressure levels for run A2.

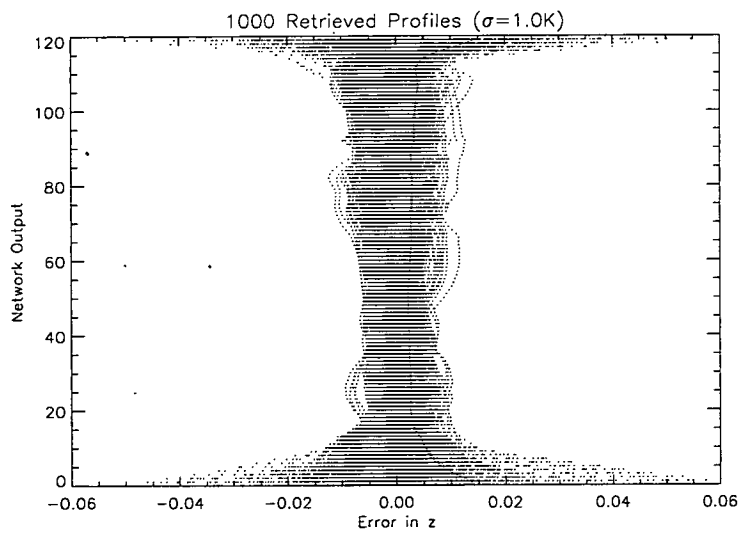

Figure 4.12: Retrieved tangent pressure levels for run B2.

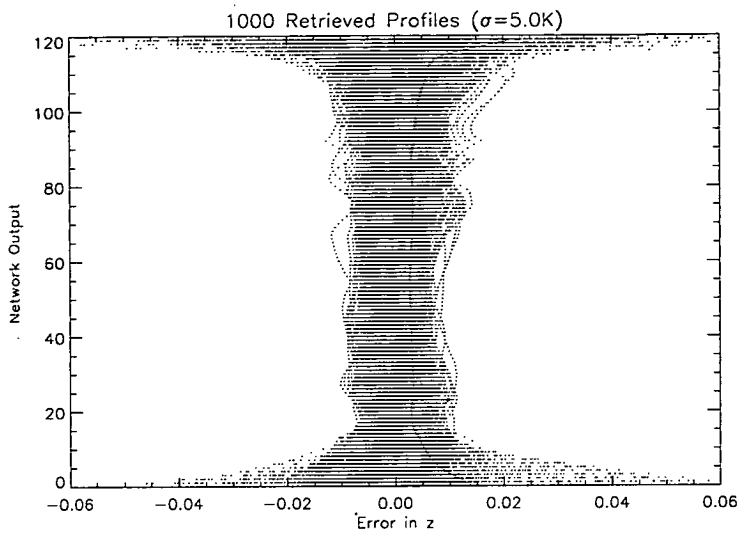

Figure 4.13: Retrieved tangent pressure levels for run C2.

instrument noise levels are also sufficiently low (around $\sigma = 0.4$ K) that training using noisy radiances isn't worthwhile in this case.

4.7 Training a network with tangent pressure levels

In the previous section, it was shown that it is possible to retrieve tangent pressure levels using a neural network. In this section, using tangent pressures as inputs to the neural-network-based forward model is examined and shown to be effective. This is done in two stages. In the first stage, the neural network is trained using pre-computed tangent pressures. In the second stage, the tangent pressures retrieved by a neural network are used.

Initially, the precomputed tangent pressures were used in the neural network. This ensures that the network can act as a forward model using accurate tangent pressures before adding in potential errors from retrieved tangent pressures. The network used is the same as the one introduced in chapter 3 with an additional 120 inputs - the tangent pressure levels - making 193 total inputs. The results from one training run can be seen in figure 4.14, which has the same format as those presented in chapter 3. As can be seen, the worst error is approximately 0.15 K, well below the instrument noise of the channel (0.37 K). This demonstrates that

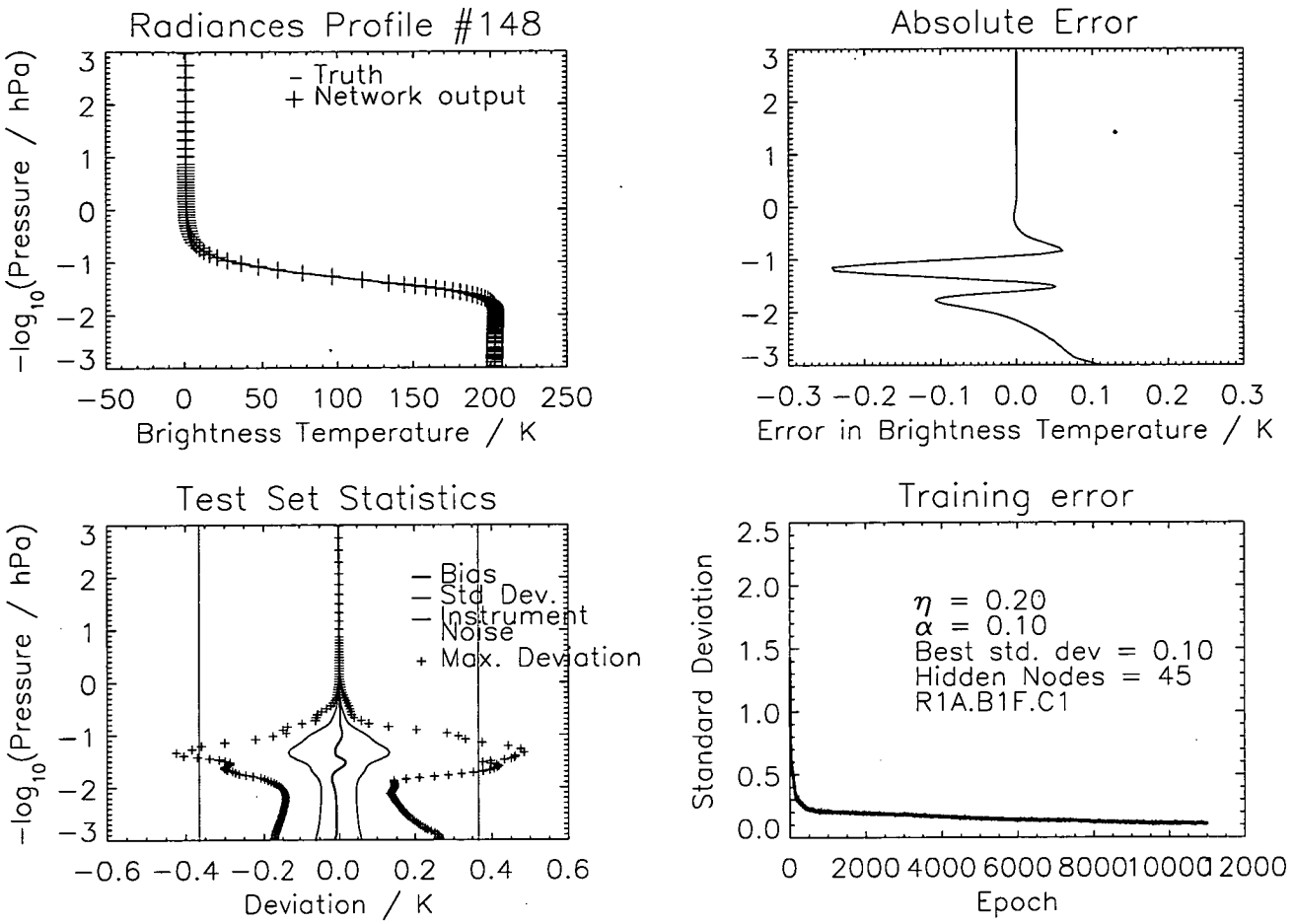

Figure 4.14: A sample neural network training run using varying tangent pressures (format is the described in section 3.5.1). The largest error is approximately 0.15K, much less than the instrument noise level but the training time has increased substantially.

if exact tangent pressures were available, it is feasible for a neural network to utilise these in a forward model.

The second set of tests examines the effect of adding noise, from the retrieval procedure, to the tangent pressures when using them as inputs to the neural network forward model. In section 4.5.2, the retrieved tangent pressures had an error with standard deviation between $\sigma = 0.002$ (in the middle of the profile) and $\sigma = 0.005$ (near the bottom).

The network that was trained on precomputed tangent pressures was reused in this process. First, the network was run with a set of 2000 input profiles to

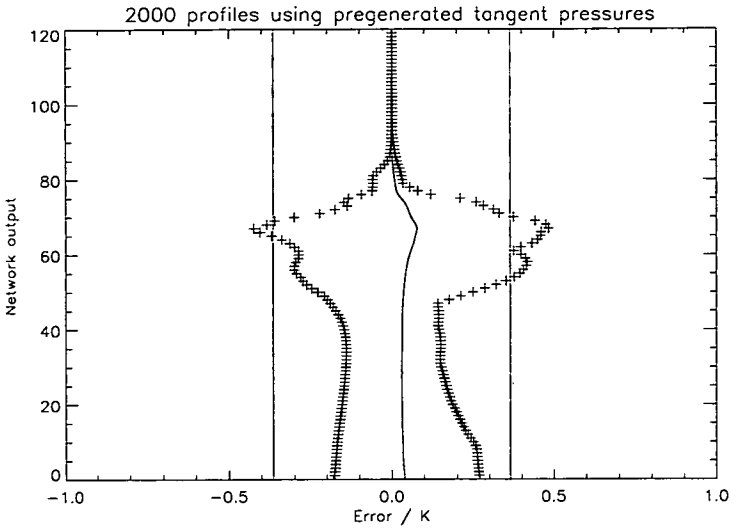

Figure 4.15: A comparison of 2000 radiances generated using a neural network compared to the Pumphrey (2006) forward model, using clean tangent pressures in both cases. The solid black line is the standard deviation of the error and the stars indicate the maximum deviation at each network output. The red lines are the expected instrument noise

generate a base for comparisons. The same network was then reused with the same temperature profiles, but with noise added to the tangent pressures. The noise was randomly generated with a normal distribution with the level of the noise varying with minor frame number. These noise levels came from the data given by figure 4.8.

The results of these runs are given in figure 4.15 for the no-noise run and figure 4.16 for the run with noise. Here, the central line gives the standard deviation through the outputs (compared to the radiances generated using a traditional forward model with no noise on the tangent pressures), the red lines indicate the noise level of the instrument and the stars indicate the maximum deviation for each network output. The aim is to keep the standard deviation within the instrument noise, as discussed in chapter 3. As can be seen, although adding noise to the tangent pressures does increase the error in the system (previously this error was around $\sigma = 0.1$ K, with noise it is around $\sigma = 0.2$ K), it is still below the instrument noise. This indicates that the network can successfully handle noise on the tangent pressures.

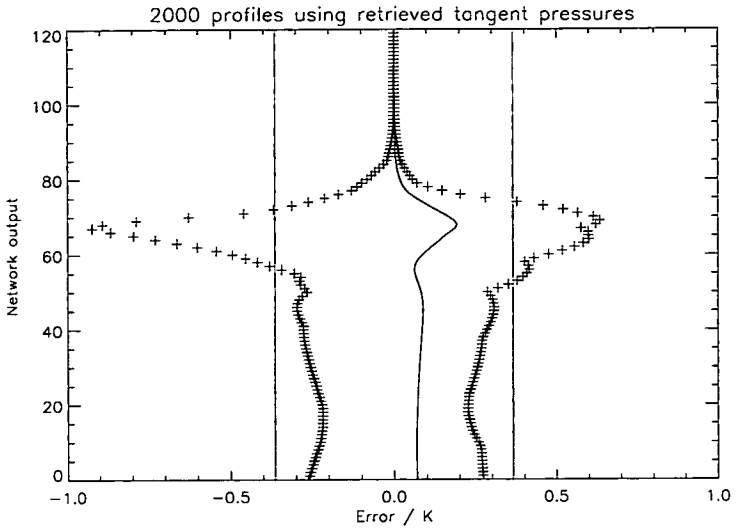

Figure 4.16: A comparison of 2000 radiances generated using a neural network with noisy tangent pressures compared to the Pumphrey (2006) forward model using clean tangent pressures. The solid black line is the standard deviation of the error and the stars are the maximum deviation at each network output. The red lines are the instrument noise. Compared to figure 4.15, there is an increased level of noise in the outputs however they are still within instrument noise levels.

4.8 Discussion

This chapter has discussed the problems posed by needing to know the tangent pressures to the process of assimilating radiances. To assimilate EOS-MLS radiances directly, tangent pressures would be dealt with in one of two ways. The preferred method would be to incorporate them into the assimilation model's state vector, which cannot be done in this case due to technical difficulties. The second way of dealing with tangent pressures would be to do a retrieval outside the assimilation scheme. Using optimal estimation methods would consume a large amount of computer power and introduces additional a-priori information into the assimilation system. Therefore a new approach must be considered if EOS-MLS radiances are to be assimilated. This approach should ideally be computationally quick to run, accurate and not introduce any a-priori information.

One proposed solution to the problem of determining tangent pressures involves using a neural network to retrieve tangent pressures from given radiances. This is what has been discussed in detail in the latter part of this chapter. It has been shown that a neural network is capable of retrieving tangent pressure

information from radiances within reasonable errors. Further, this network is as been shown to be able to deal with noisy radiances.

Overall, it is felt that using a neural network to retrieve tangent pressure information is a strong possibility in a real system. It is much faster than traditional methods, can be used independently of any retrieval system and does not include any a-priori information on what the tangent pressure profile will look like.

Chapter 5

Extending the Neural Network

5.1 Introduction

This chapter looks at ways of extending the neural network-based forward model introduced previously. The networks described so far have simulated a simplified model as a proof of concept. In chapter 3 the network took only a temperature profile, measured at fixed pressure levels, and produced a set of radiances, again at fixed pressure levels, for a single channel in a single band. Chapter 4 extended the neural network to include varying tangent pressure levels, creating a more realistic model.

This chapter generalises the neural network to a more complete model. This is done in two ways. Firstly the network is extended by including more channels within band 1. The second way of extending the neural network is to use chemical species within the calculation to give a more realistic forward model.

5.2 Extending the Network to More Channels

The EOS-MLS instrument includes 1237 channels spread across 34 bands. The distribution of these channels can be found in table 2.1 in chapter 2. Up until now, the neural network has been concentrating on a single channel (R1A.B1F.C1). It was decided that the neural network should first be extended to deal with additional channels within band 1 before attempting to add extra chemical species.

Satellite data used in assimilation processes are generally limited to temperature and ozone measurements. As was previously mentioned, band 1 of the

EOS-MLS is centered on an strong oxygen line and effects from other species are negligible in comparison. For this reason, only temperature need be considered when running most channels in band 1 in the neural network forward model.

5.2.1 The Network Architecture

In order to keep the network to a manageable size, it was decided that each channel should be modelled using a separate neural network. This has several advantages, namely that the problem can then be considered, in the jargon phrase, *embarrassingly parallel* and also reduces required complexity of each network.

An embarrassingly parallel problem is one that each subprocess can be run independently and hence be sent to a separate processor in a computer system. In this case, the complete system can be considered a doubly embarrassingly parallel system as each neural network can be considered a separate process and then each node within the network can be considered a separate process (albeit depending on inputs from previous layers of nodes).

All the networks in this section trained here have 193 inputs, 73 inputs representing the temperature profile and 120 inputs representing the tangent pressure levels, and the 120 outputs, representing the radiance profile. As before, these inputs and outputs are normalised according to equations 3.5 and 3.6 in chapter 3. Each network was started with 45 hidden nodes - the number found in section 4.7 of chapter 4 to produce the best results in channel 1 - with this number being altered heuristically between training runs.

5.2.2 Training

Training was carried out in a similar way to previously. Several networks were trained for each channel, with the number of hidden nodes, the learning rate and momentum being altered between runs.

The results of the best run for each channel can be found in table 5.1. As can be seen, almost all channels meet the requirement of being within instrument noise and several are below half the instrument noise. This shows that the network can be extended to include a full band.

Channel	*Instrument Noise / K*	*Network error / K*	*Hidden Nodes*
1	0.37	0.13	45
2	0.37	0.15	45
3	0.37	0.18	45
4	0.34	0.25	45
5	0.34	0.25	40
6	0.34	0.26	45
7	0.33	0.28	45
8	0.33	0.26	38
9	0.33	0.27	45
10	0.32	0.22	60
11	0.32	0.29	55
12	0.32	0.16	40
13 *	0.32	0.70	45
14	0.32	0.20	50
15	0.32	0.27	70
16 *	0.32	0.31	55
17 *	0.33	0.32	40
18 *	0.32	0.31	55
19	0.32	0.27	60
20	0.33	0.21	50
21	0.33	0.21	45
22	0.32	0.27	60
23	0.34	0.24	45
24	0.33	0.15	45
25	0.33	0.19	45

Table 5.1: A list of channels in Band 1 giving the instrument noise level, the validation error of the network and the number of hidden nodes in the network for the best training run. Channels marked with * are not considered well trained and are looked at in detail in section 5.2.3.

5.2.3 Badly Trained Channels

There are several channels that are not trained to within instrument noise or are very close to instrument noise - channel 13 is well out-with this target and channels 16, 17 and 18 are very near the limit. This section will look at why these channels are not well trained and ways of improving the training of these channels.

Figures showing the trained network output for channels 16, 17 and 18 can be found in figures 5.1, 5.2 and 5.3. As can be seen in figures 5.1 and 5.2, in these cases the problems occur near the "transition phase" of the profile - the phase of the radiance profile where the radiance grows from no signal (around $z = 1.0$ in figure 5.1) to saturated (around $z = -0.5$ in figure 5.1) and the instrument is receiving most information about the atmosphere and where we want the best results. Figure 5.3 shows that the bias in channel 18 is very large around the transition phase. This implies that in these channels, the training data may be insufficient, resulting in badly trained networks. This effect may be mitigated due to changes in the operating specifications of the instrument, discussed below.

The Newer Training Data

During the course of this research, the instrument specifications were changed slightly. Previously, the instrument was designed to scan 120 minor frames per major frame. The updated instrument specifications increased this to 125 minor frames per major frame. In addition to this, a higher resolution temperature set was developed for use with the forward model. This new set of temperature profiles extends between $-3 \leq z \leq 5$ (1000hPa to 0.00001hPa) instead of between $-3 \leq z \leq 3$ (1000hPa to 0.001hPa). This increased the number of temperature points in a profile from 73 to 97. The temperature set is also at a higher resolution in the upper atmosphere (above $z = 0$), allowing radiances in the upper region of the atmosphere to be much more accurately determined.

Using this new data, new training, validation and testing sets were constructed. The training set was extended to 2400 profiles from 1500 while the validation and testing sets were kept the same size - 300 and 200 profiles respectively. The number of network inputs was increased to 222, 125 tangent pressures and 97 temperatures, and 125 radiance outputs.

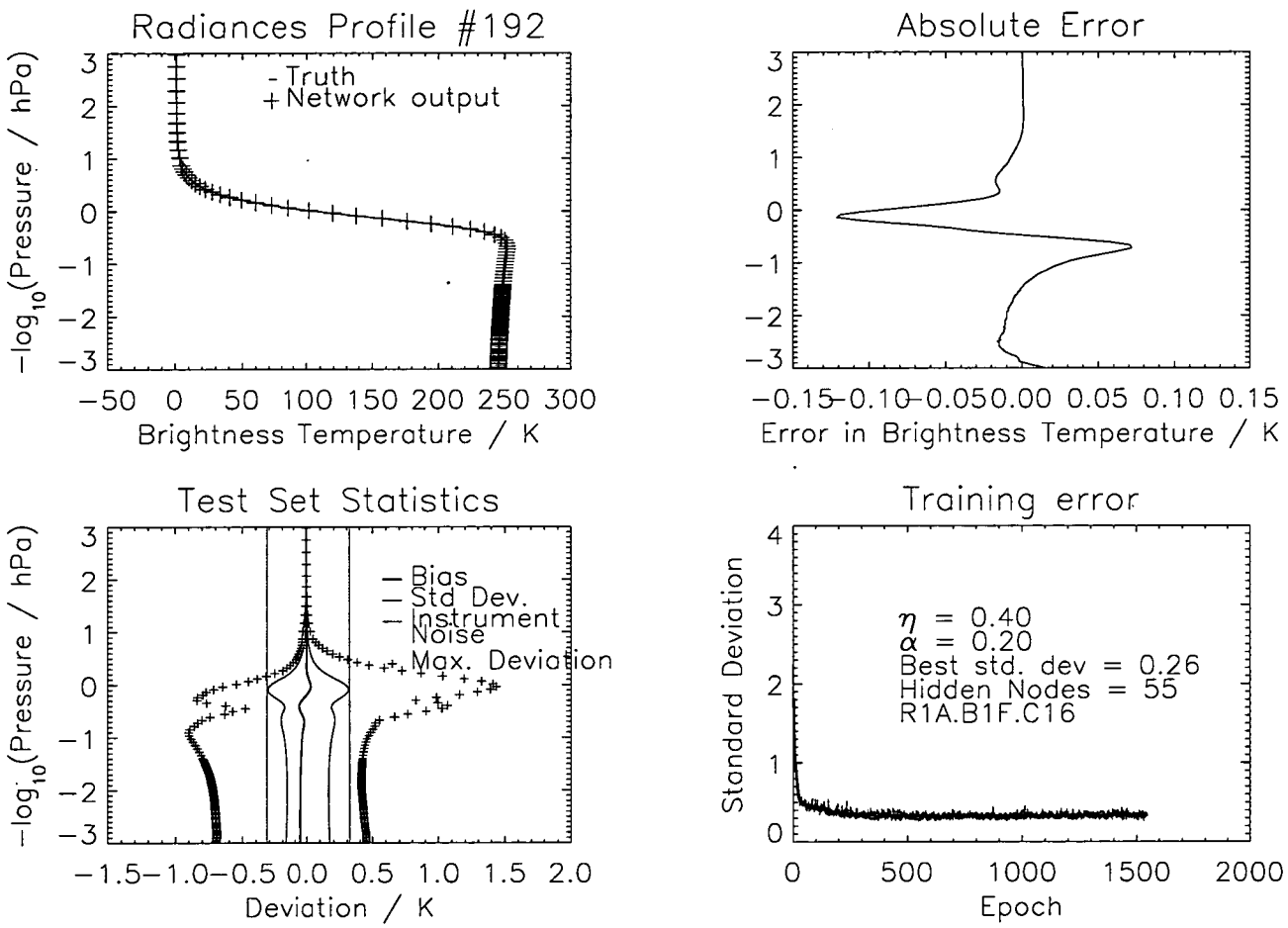

Figure 5.1: A neural network training run for channel 16 of band 1 (format is the described in section 3.5.1). The format of the figure is the described in section 3.5.1 of chapter 3. Here, the error on the test set is $\sigma = 0.31$ K, very close to the instrument noise level in this channel ($\sigma = 0.32$ K).

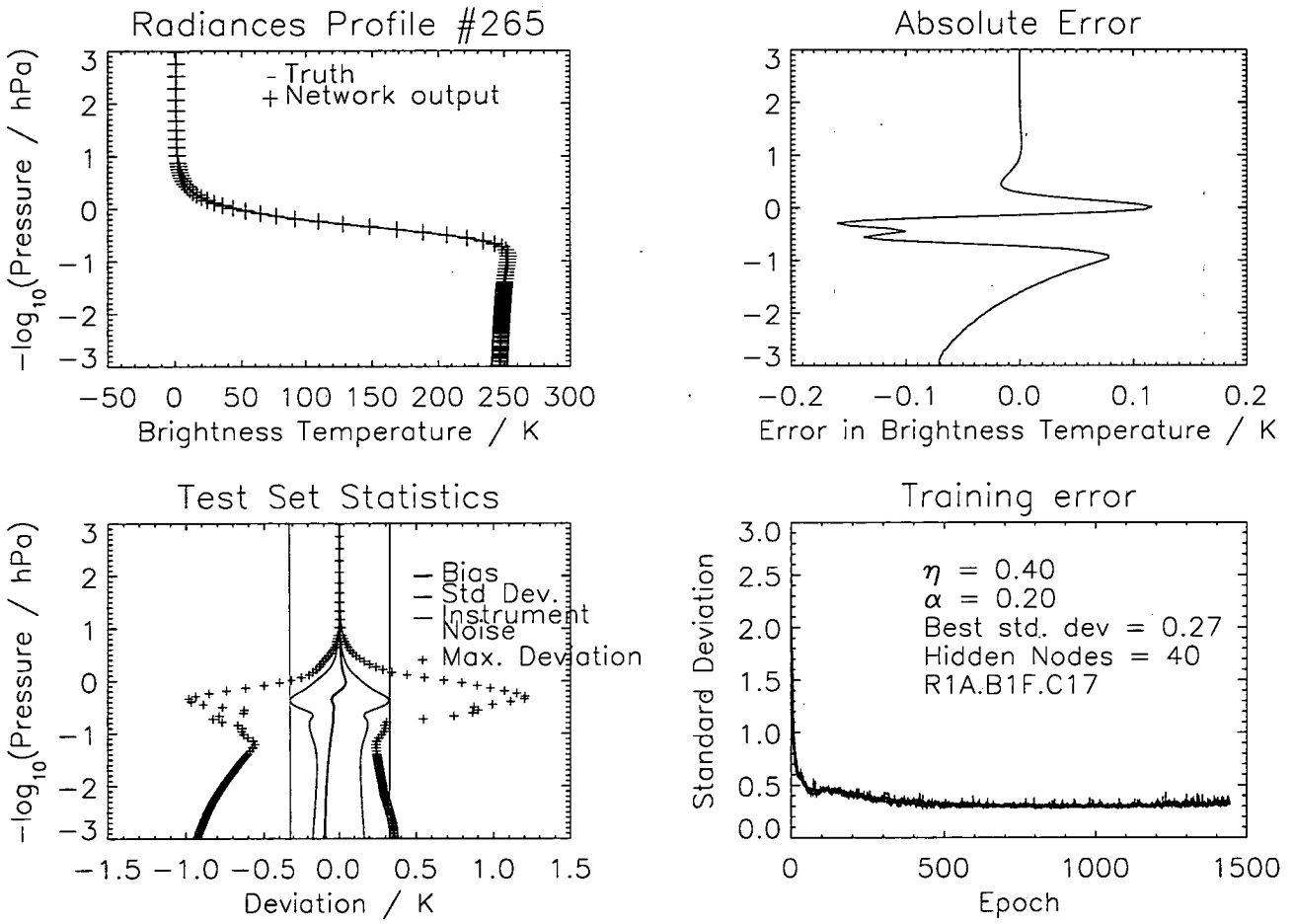

Figure 5.2: A neural network training run for channel 17 of band 1 (format is the described in section 3.5.1). Here, the error on the test set is $\sigma = 0.32$ K, very close to the instrument noise level in this channel ($\sigma = 0.33$ K).

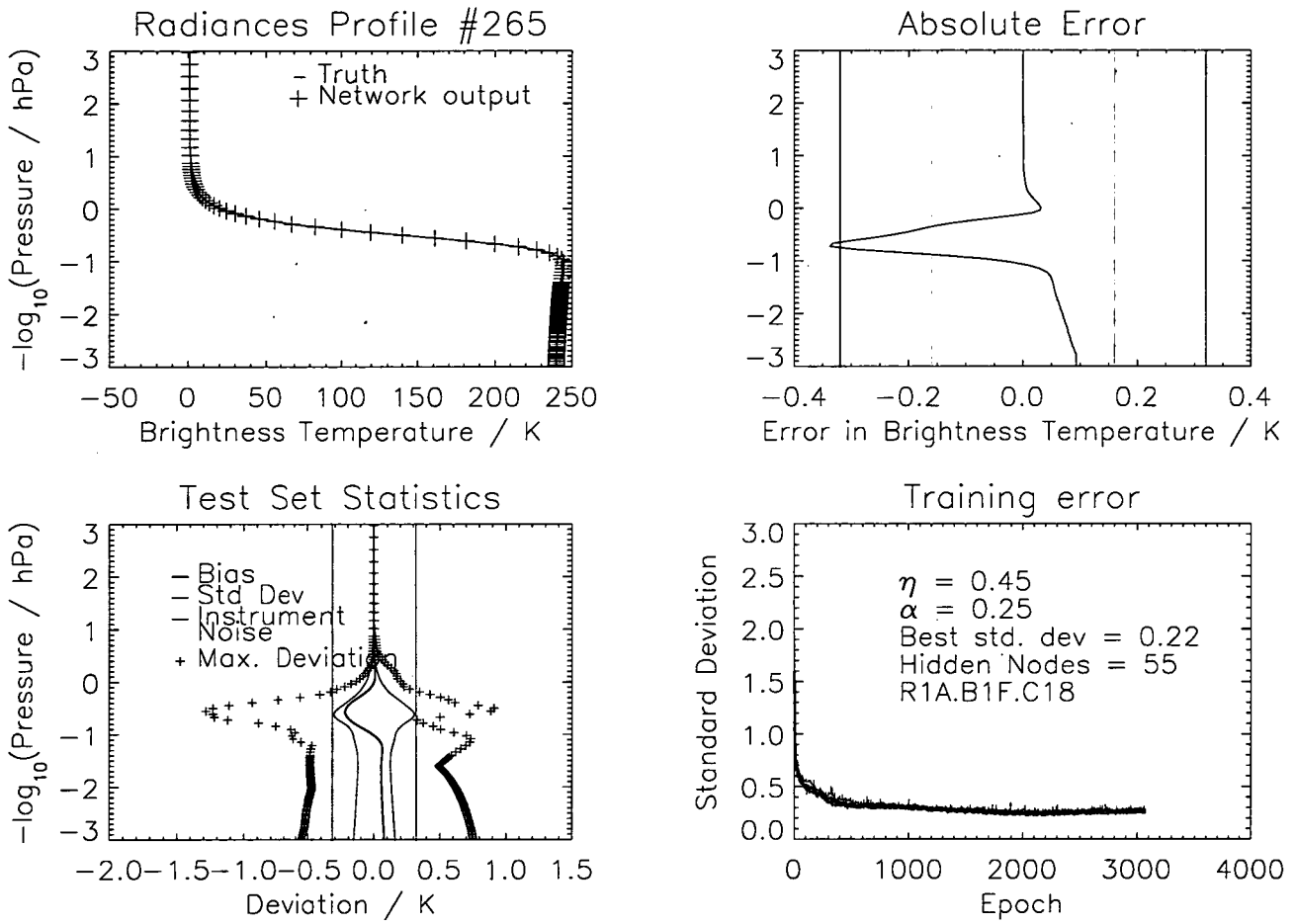

Figure 5.3: A neural network training run for channel 18 of band 1 (format is the described in section 3.5.1). Here, the error on the test set is 0.31 K, very close to the instrument noise level in this channel ($\sigma = 0.32$ K).

5.2.4 The Neural Network With the Updated Training Sets

Using the updated training sets, channels 16, 17 and 18 were retrained. The results from the best training runs for each channel can be seen in figures 5.4, 5.5 and 5.6, with a summary presented in table 5.2. As can be seen, the test set errors are now well below the required noise levels of the instrument.

Channel	*Instrument Noise / K*	*Old Network Error / K*	*Final network error / K*	*Hidden nodes*
16	0.32	0.31	0.21	50
17	0.33	0.32	0.28	50
18	0.32	0.31	0.17	65

Table 5.2: A list of channels that had difficulty previously with their new validation errors

The final channel that caused problems in the original neural network was channel 13, the central channel in band 1. The original network output can be seen in figure 5.7 where the change to the scaling of the y-axis should be noted. Previously, the first three graph's y-axes cover the range $z = [-3, 3]$ as anything above $z = 3$ can be assumed to come from background radiation. In channel 13 however, this is not the case and here the y-axis of these plots has been extended to cover the range $z = [-3, 4]$. The fact that channel 13 receives a signal above $z = 3$ also provides a possible explanation of why this channel performs so badly. As can be seen in figure 5.7, the network cannot satisfactorily simulate the forward model right at the top of the profile (around $z = 3$ or 0.001 hPa). This may be due to the lack of temperature inputs at this level (the old temperature training data only extended up to $z = 3.0$) and by adding more, the problem may be better handled by the network. As was mentioned in section 5.2.3, the new datasets used for training extend the temperature profile much higher than previously (up to $z = 5.0$). Using this new data, a network was trained and produced the results shown in figure 5.8.

As can be seen, all the network outputs are now well within the noise level, with a network error level of 0.13 K (Instrument noise in this channel is 0.32 K). There is still a large bias which can be removed from the results of the network in operation, further improving accuracy.

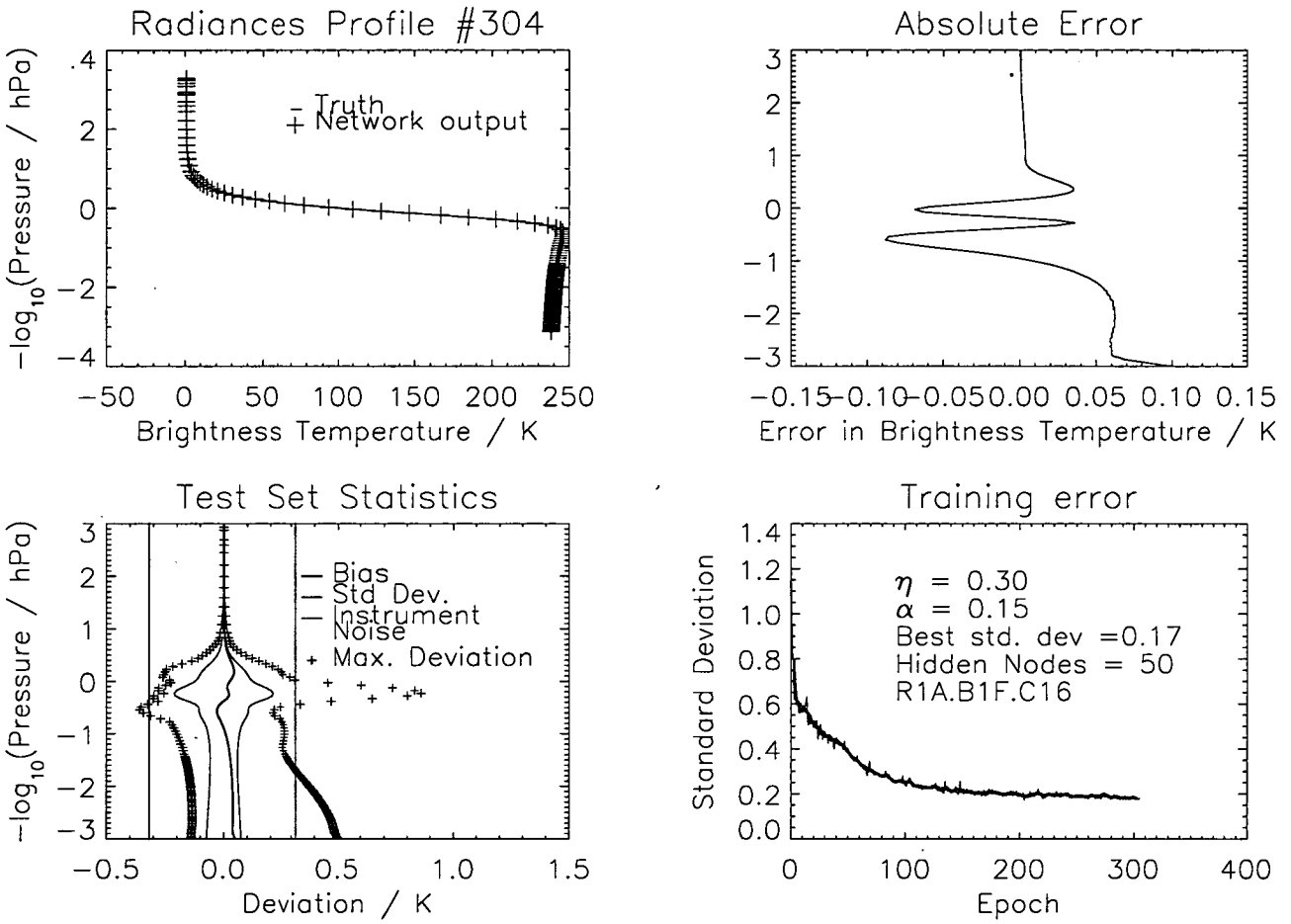

Figure 5.4: A neural network training run for channel 16 of band 1, using the new dataset (format is the described in section 3.5.1). Here, the error on the test set is now 0.211 K, much lower than previously (figure 5.1)

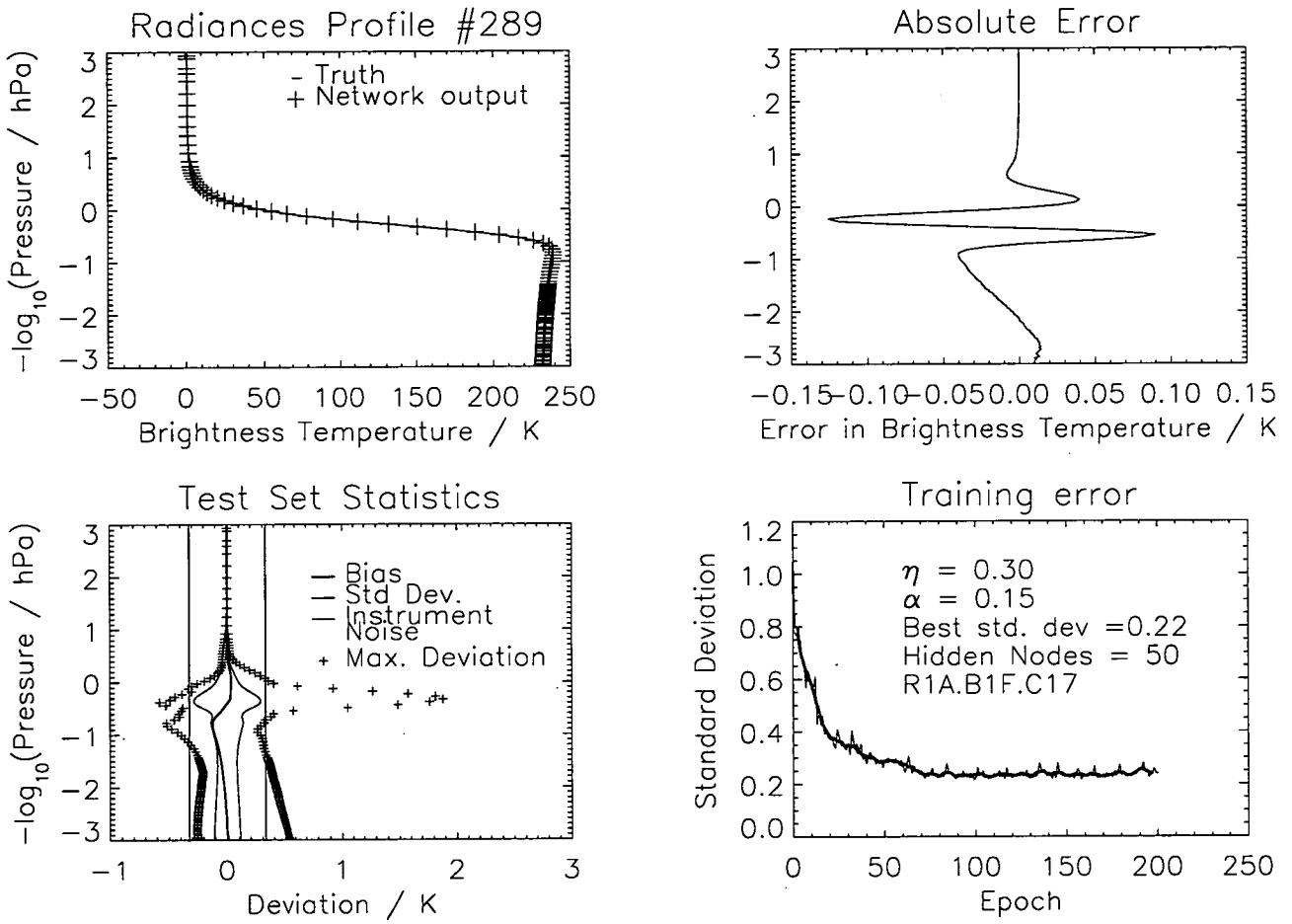

Figure 5.5: A neural network training run for channel 17 of band 1, using the new dataset (format is the described in section 3.5.1). Here, the error on the test set is now 0.284 K, much lower than previously (figure 5.2)

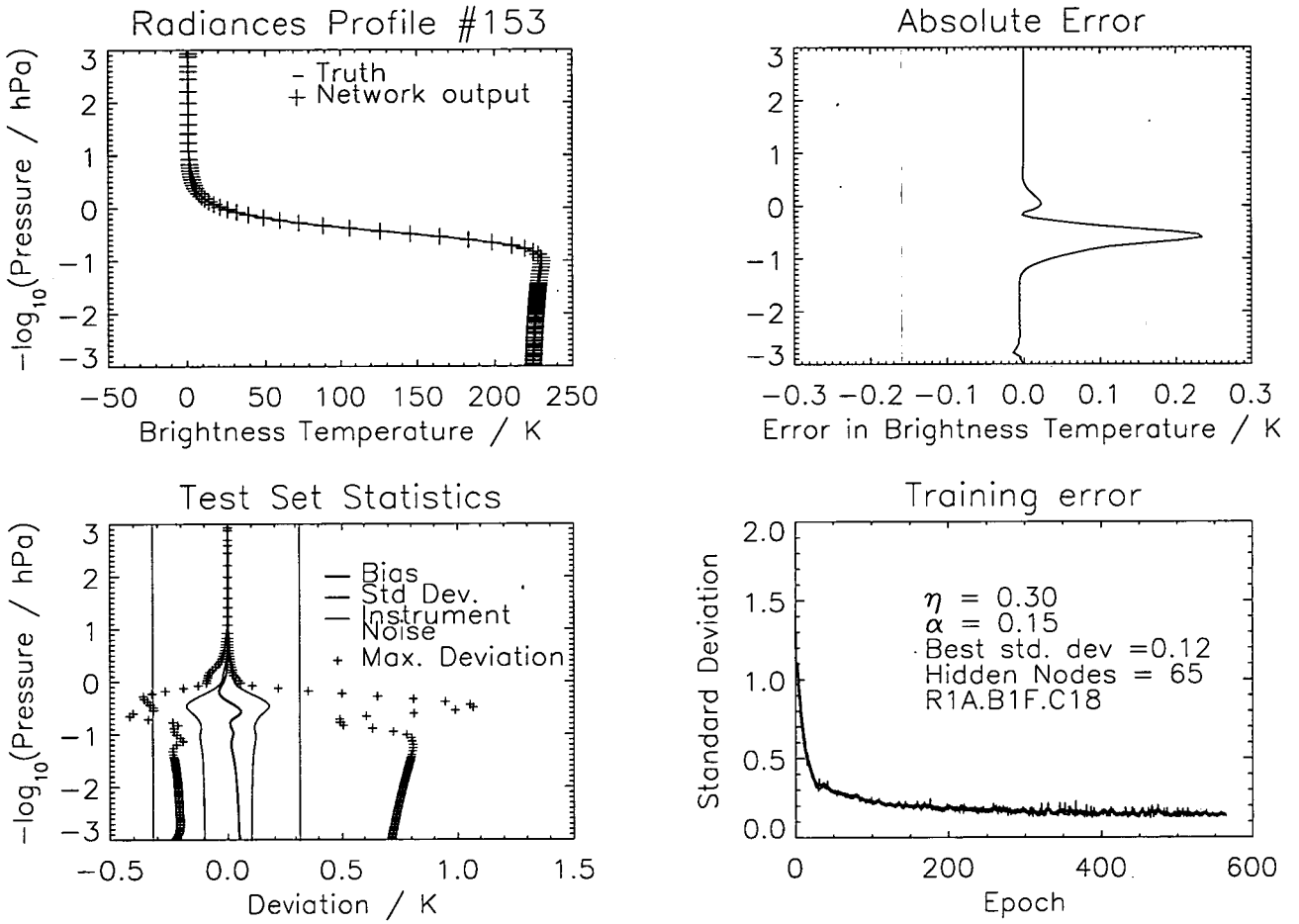

Figure 5.6: A neural network training run for channel 18 of band 1, using the new dataset (format is the described in section 3.5.1). Here, the error on the test set is now 0.176 K, much lower than previously (figure 5.3)

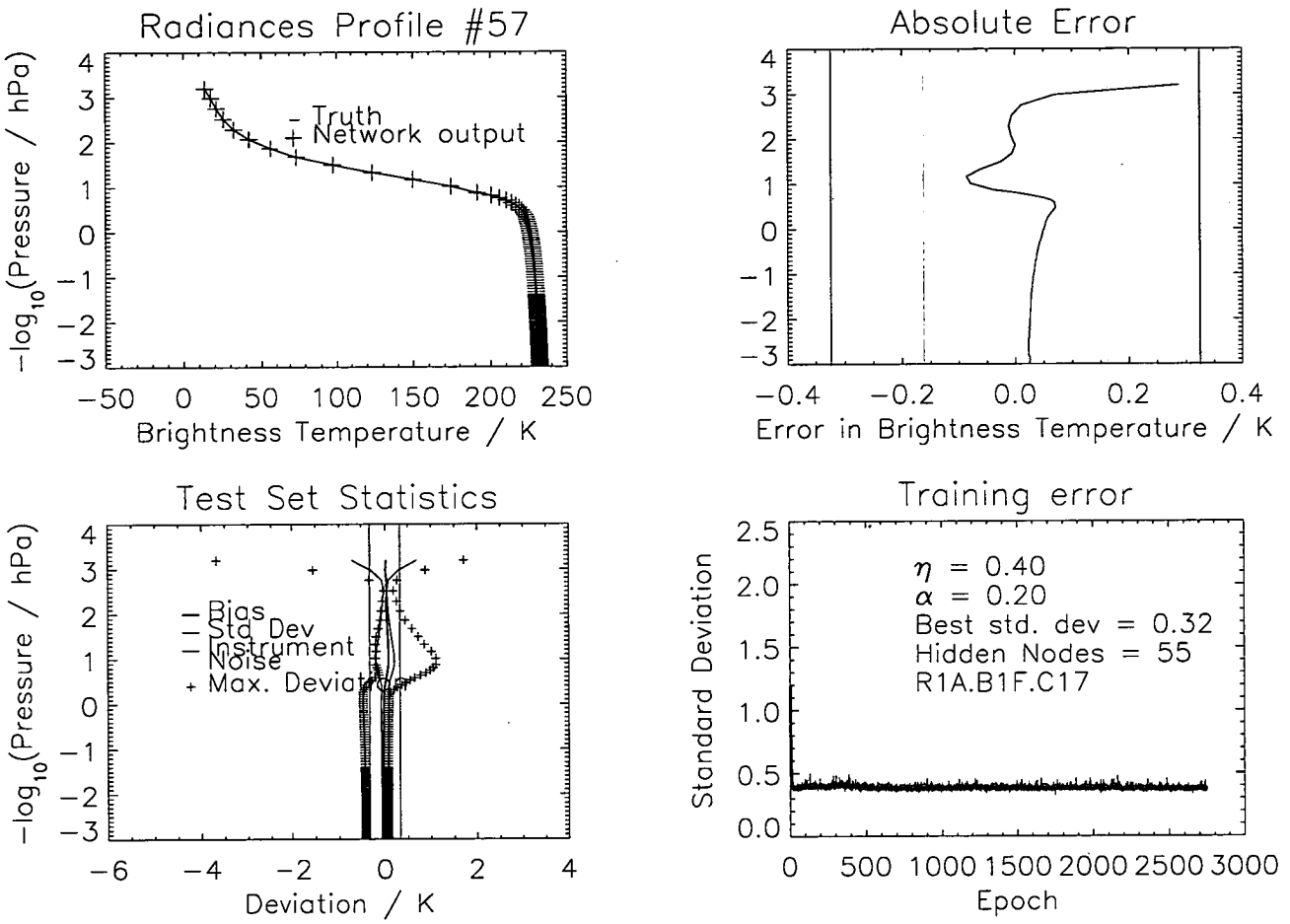

Figure 5.7: A training run for channel 13 of band 1 using the old dataset (format is the described in section 3.5.1). The y-scale of the first 3 sub-diagrams have been extended to [-3,4] as the knee of the profile extends up past 3, as was previously used. The error in the test set is 0.698 K, which is clearly unacceptable.

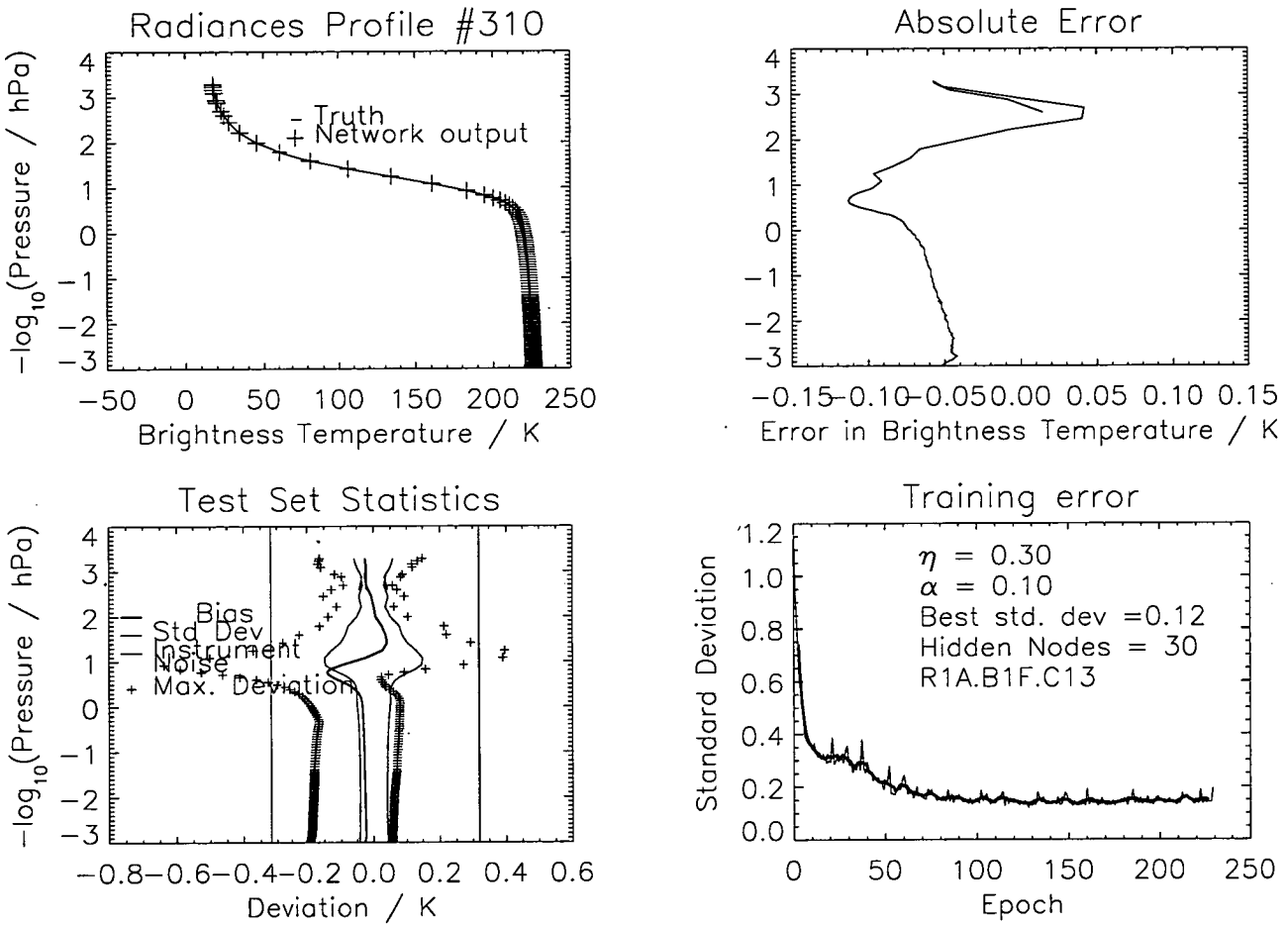

Figure 5.8: A training run for channel 13 of band 1 using the new dataset (format is the described in section 3.5.1). The error now has been reduced to 0.13 K, much improved from previous training runs (figure 5.7).

5.3 Dealing With Chemical Species

So far, all the neural networks examined here have been trained using only temperature profiles. That is, the training data have been generated assuming that only oxygen and temperature have an effect on radiances. This is clearly a large simplification. In reality, there are a large number of chemical species in the atmosphere which absorb and emit in the measured bands and hence need to be taken into account, even for temperature assimilation. This section looks at how this can be done for a neural network forward model.

In the assimilation model, only a few species are present. The majority of bands on the EOS-MLS instrument are centered on spectral lines that are not in the assimilation model and so are not useful in the assimilation at the moment. There are several bands that would be useful in an assimilation scheme but have signals from chemical species that are not part of the assimilation. For these bands, the forward model must account for these additional species while the assimilation process must supply a profile for these additional species, to accurately simulate the radiances and save contamination of other information, useful to the assimilation.

Here, the effect of chemical species are considered on two bands: band one and band seven. Band one is the band considered previously and is centered on a strong oxygen line. The effects of chemical species in this case is small. Band seven is a band centered on the 235.7 GHz ozone line and as such is strongly influenced by this. Here, only two chemical species are considered: ozone (O3) and water vapour (H2O).

5.3.1 Band One

In order to assess whether it is viable to include additional species in the neural network, it is useful to investigate the effects of species on a band where these effects will be small. Band 1, the band used previously, is centered on a strong oxygen line and so the effects from other species is minimal. There are several channels, though, that are affected by minor ozone lines. In this section, the effect of these ozone lines are discussed.

Figure 5.9 shows the calculated spectrum received at the satellite as a function

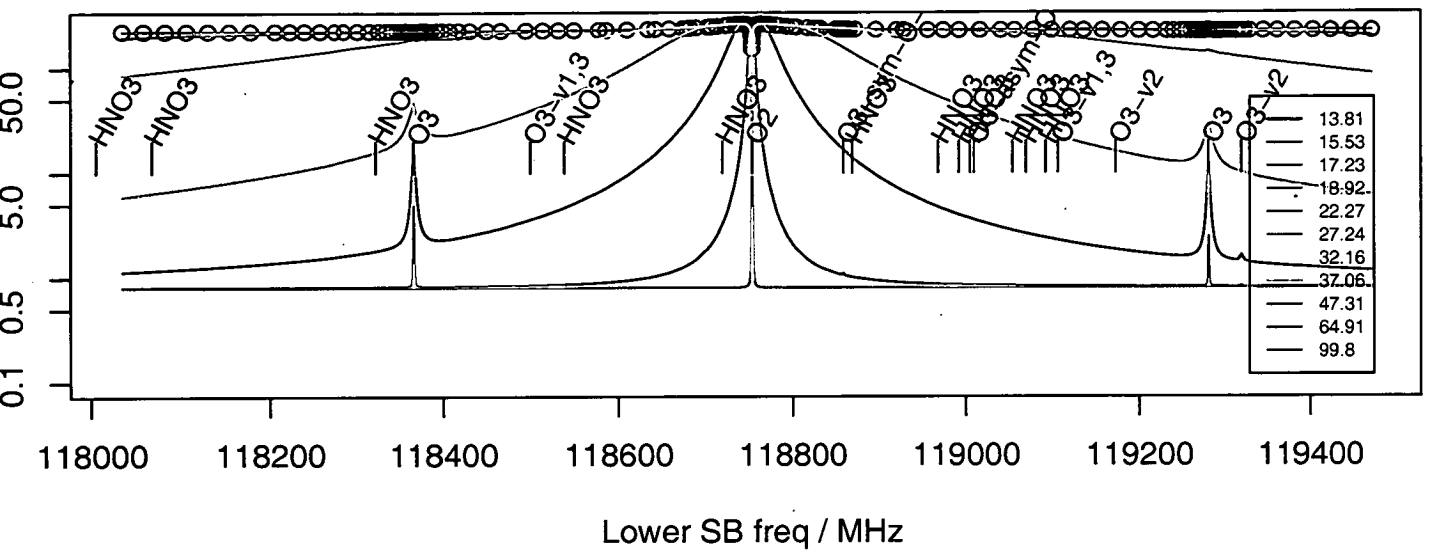

Figure 5.9: Radiance received by the EOS-MLS as a function of frequency for the frequency range of the lower sideband of band 1, calculated from an atmosphere in which O2, HNO3, O3 and H2O are significant emitters in the frequency range. The inset scale gives the tangent height of the radiance in kilometres and the circles represent the sampling points used to generate the plot.

of frequency for different tangent heights for the lower sideband of band 1[1] with O2, O3, HNO3 and H2O present in the modelled state. In addition to the central O2 spectral line, which dominates the radiances for the band, there are two O3 spectral lines - around 118.35 GHz and 119.30 GHz - which produce a measurable effect on radiances. Although water vapour doesn't have any lines centered in this region, there is water vapour continuum which effectively reducing the depth the instrument can look through.

Several radiance profiles from band 1 are given in figures 5.10, 5.11 and 5.12 which show, respectively, the original radiance profiles without any chemical species, the radiance profile with water vapour and nitric acid in the forward model and the radiance profile with ozone, nitric acid and water vapour added. In both the latter cases, the left figure shows the change in radiances with height compared to the base case of figure 5.10. As can be seen, adding nitric acid and water vapour has no significant effects on the radiance profile (differences have a magnitude of around 0.1 K) and are due almost entirely to water vapour. Adding

[1]The upper sideband is masked in this band and does not contribute to the final radiances

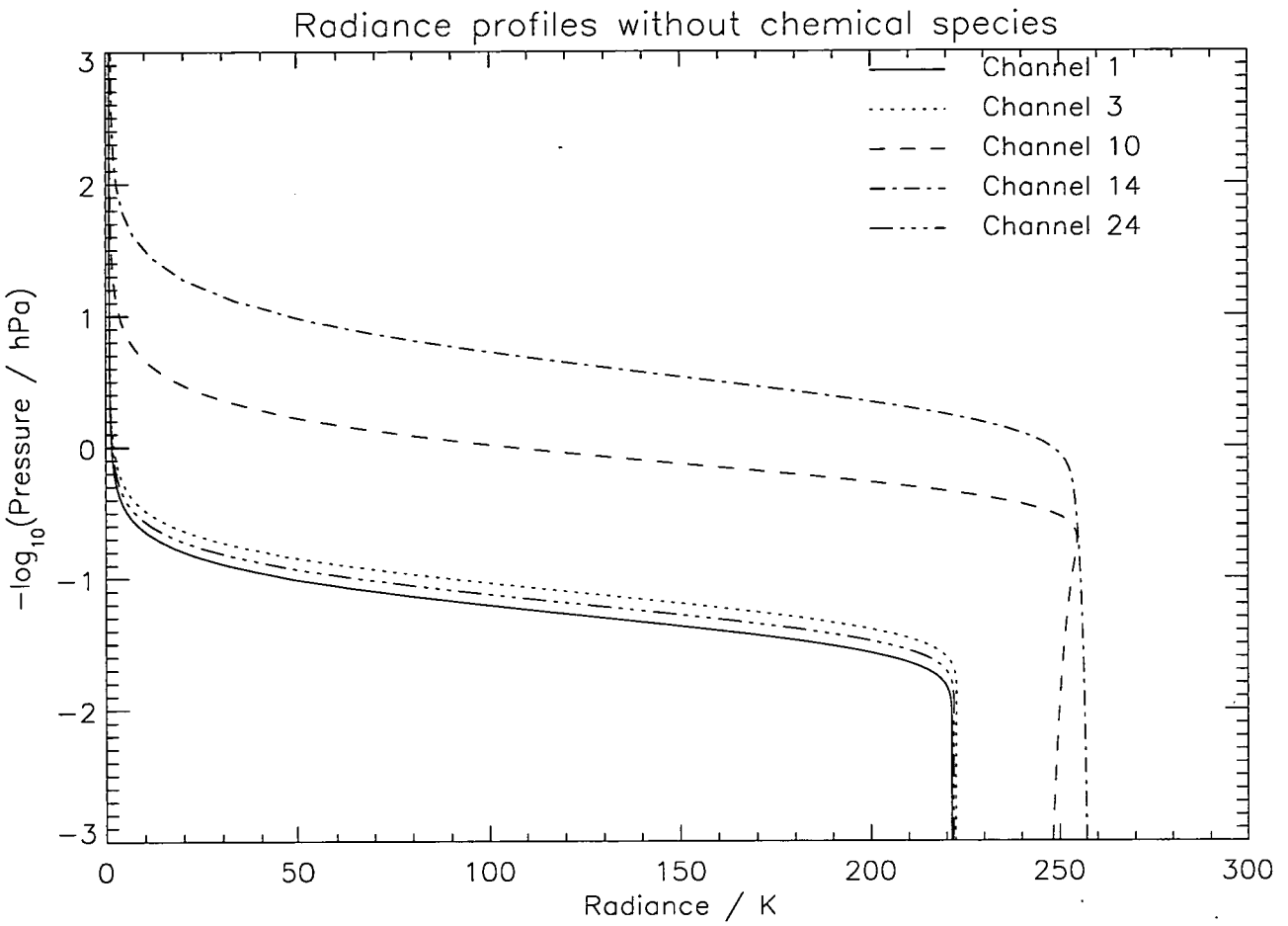

Figure 5.10: Radiance profiles for channels 1, 3, 10, 14 and 24 of Band 1 generated with no species information in the forward model.

ozone to the profiles significantly changes channels 3 and (to a lesser extent) 24. In channel 3, ozone typically increases radiances by around 10 K near 100 hPa and channel 24 increases by around 3 K in the same area. This is shown clearly in figure 5.9 where the two large ozone lines show large spikes in the radiances at the frequencies that correspond to these channels.

Having seen how chemical species affect radiances in band 1, it is necessary to try running a neural network forward model including these effects. If the neural network cannot handle the relatively small effects in band 1, other bands would have larger problems. An initial impression of the importance can be gained by modelling only one channel - channel 3. This is the channel with the most difference due to ozone included.

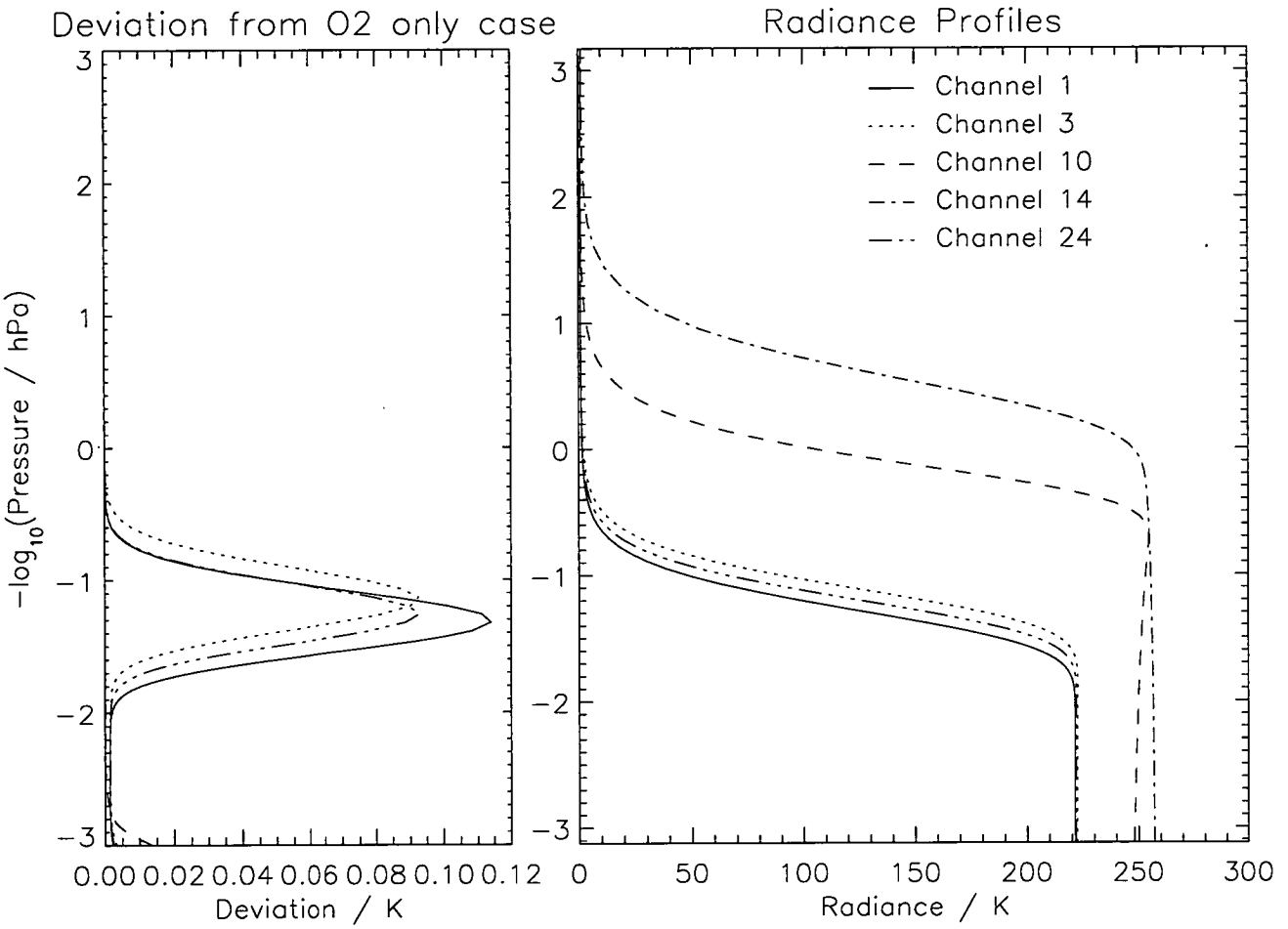

Figure 5.11: Radiance profiles for channels 1, 3, 10, 14 and 24 of Band 1 generated with water vapour and nitric acid information in the forward model. The left sub-figure shows the difference from figure 5.10. It can be seen that the maximum difference is again around 0.1 K which is almost completely due to the presence of water vapour.

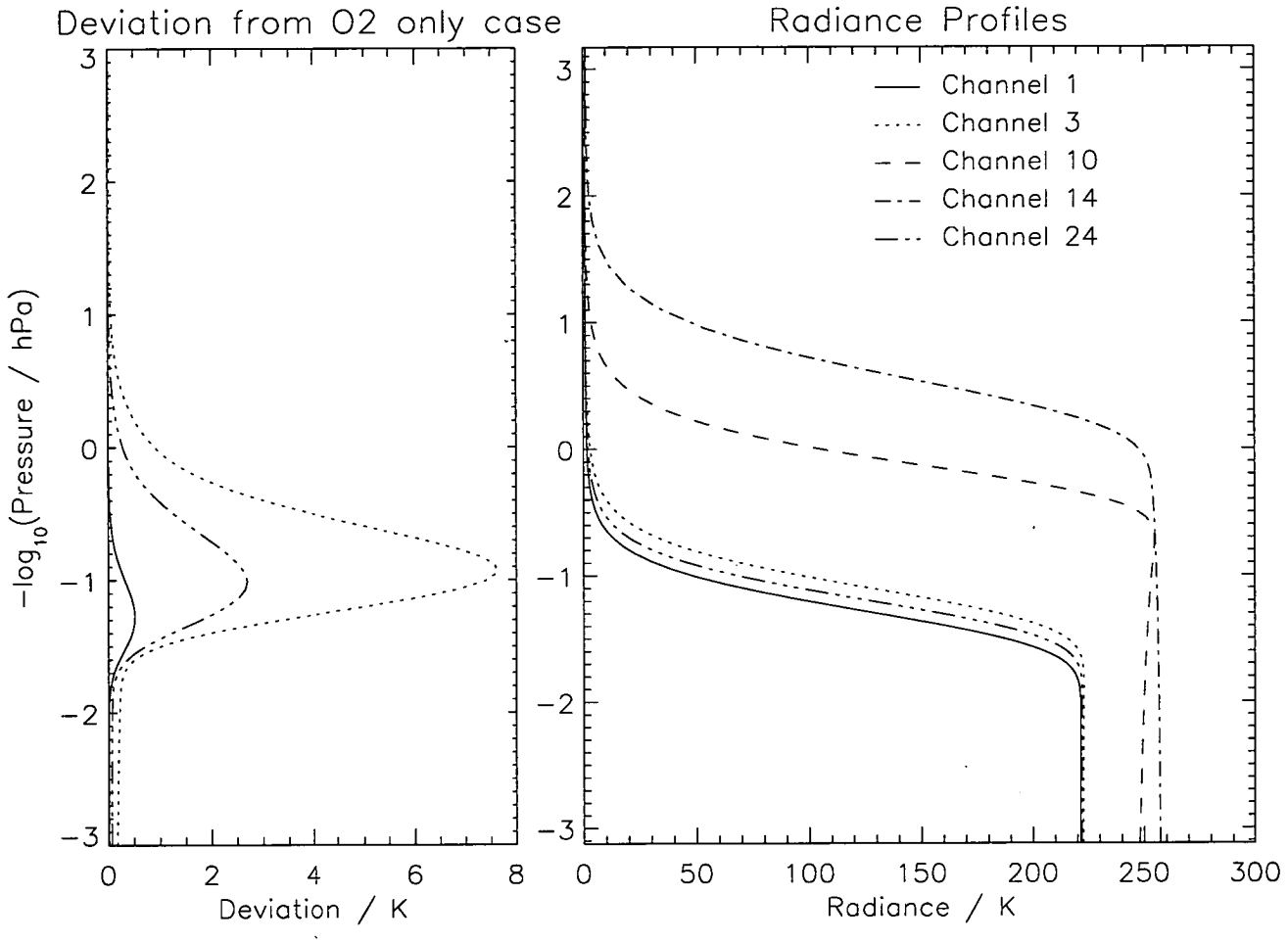

Figure 5.12: Radiance profiles for channels 1, 3, 10, 14 and 24 of Band 1 generated with water vapour, nitric acid and ozone information in the forward model. The left sub-figure show the difference from figure 5.10. Here, the largest difference is around 8 K which is in channel 3 and is caused by the large spectral ozone line there.

Results

A new training set was constructed using the same temperature and tangent pressure information as was used previously. In addition, ozone data, taken from the same source as the temperature data, was included in the forward model calculations. This ozone data covers the expected range of ozone values throughout the atmosphere.

Initially, the new training set was used to train a neural network in the same configuration as previously - 97 temperature and 125 tangent pressure inputs, and 125 radiance outputs. This was done to assess whether the ozone profile is required as an input to the neural network. Ignoring the ozone profile as an input resulted in errors of around $\sigma = 0.9$K, approximately three times the instrument noise for this channel. This shows that ozone is required as an input.

The ozone profiles are supplied as a set of 85 concentrations at fixed pressure heights. Including this information into the neural network as inputs increases the number of inputs from 222 to 306[2]. One result of this added complexity to the neural network is the requirement for more hidden nodes. Previously, 45 hidden nodes were used to train channel 3 resulting in an error of $\sigma = 0.18$ after training. Now, 120 hidden nodes are needed in order to train the network properly. The results of one training run with ozone included can be found in figure 5.13. In this case, the error is around $\sigma = 0.24$ K, which is still lower than the instrument noise thus ozone can be handled in this case.

5.3.2 Band Seven

The previous section showed that ozone could be dealt with in band 1, which is centered on a strong oxygen line. The instrument has several bands centered on ozone lines and in order to assimilate ozone information from the EOS-MLS, a forward model is required for these bands. One of these is band 7, a band that it is highly nonlinear, creating much more work for the forward model.

Example radiance profiles from several channels in band 7 are shown in figure 5.14, which were generated using temperature, ozone and water vapour as species input. As can be seen, these profiles are very different from the profiles from band 1. The addition of water vapour in this band causes changes of the

[2]One ozone input is constant across all profiles and so can be considered as part of the bias node

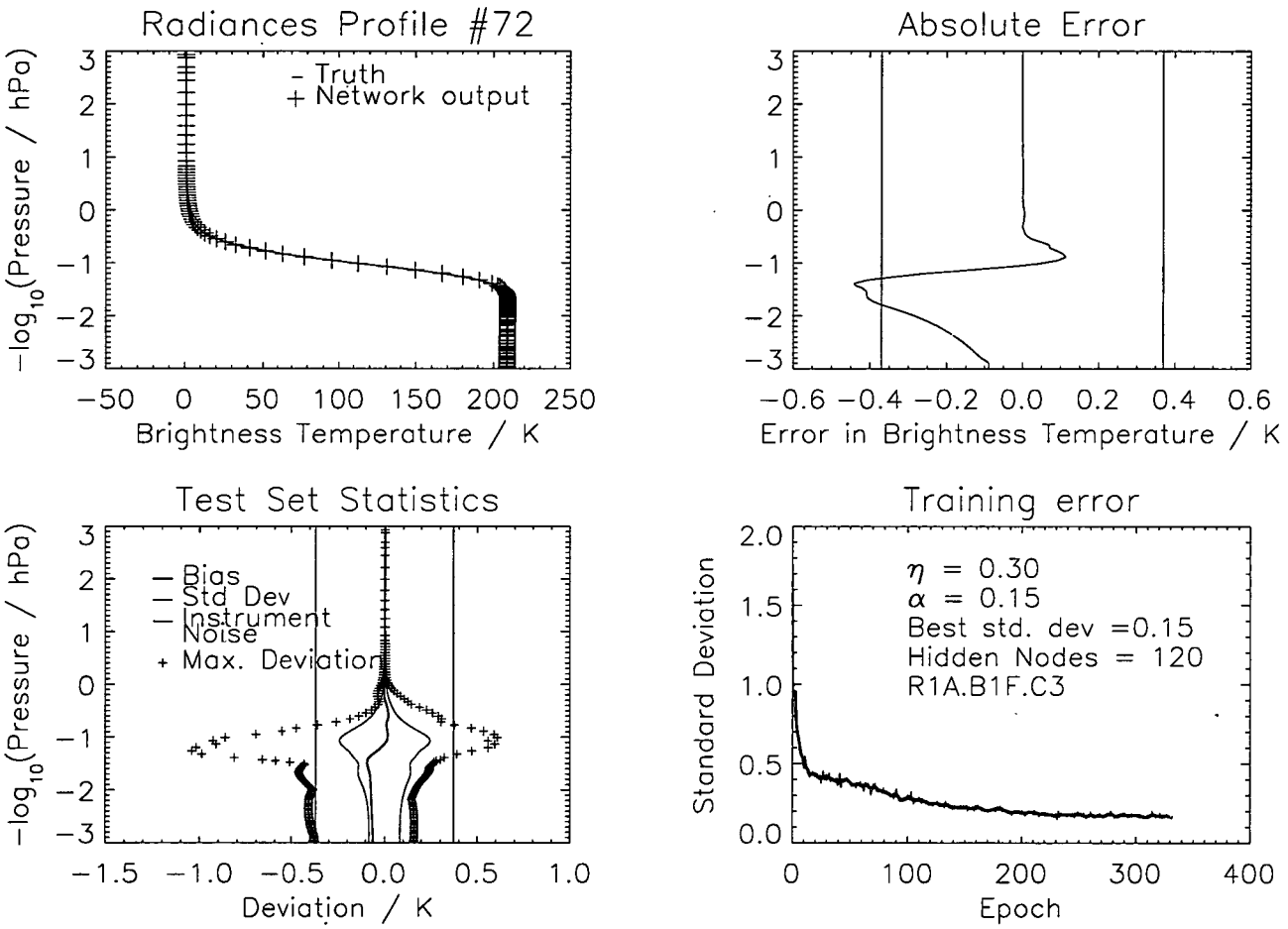

Figure 5.13: Training a network for channel 3 of band 1 with ozone (format is the described in section 3.5.1). Here, ozone concentration is used as part of the inputs to the neural network. The error in the network during testing is around 0.24 K, well below the instrument noise (0.37 K). This shows that a neural network can cope with chemical species in this context.

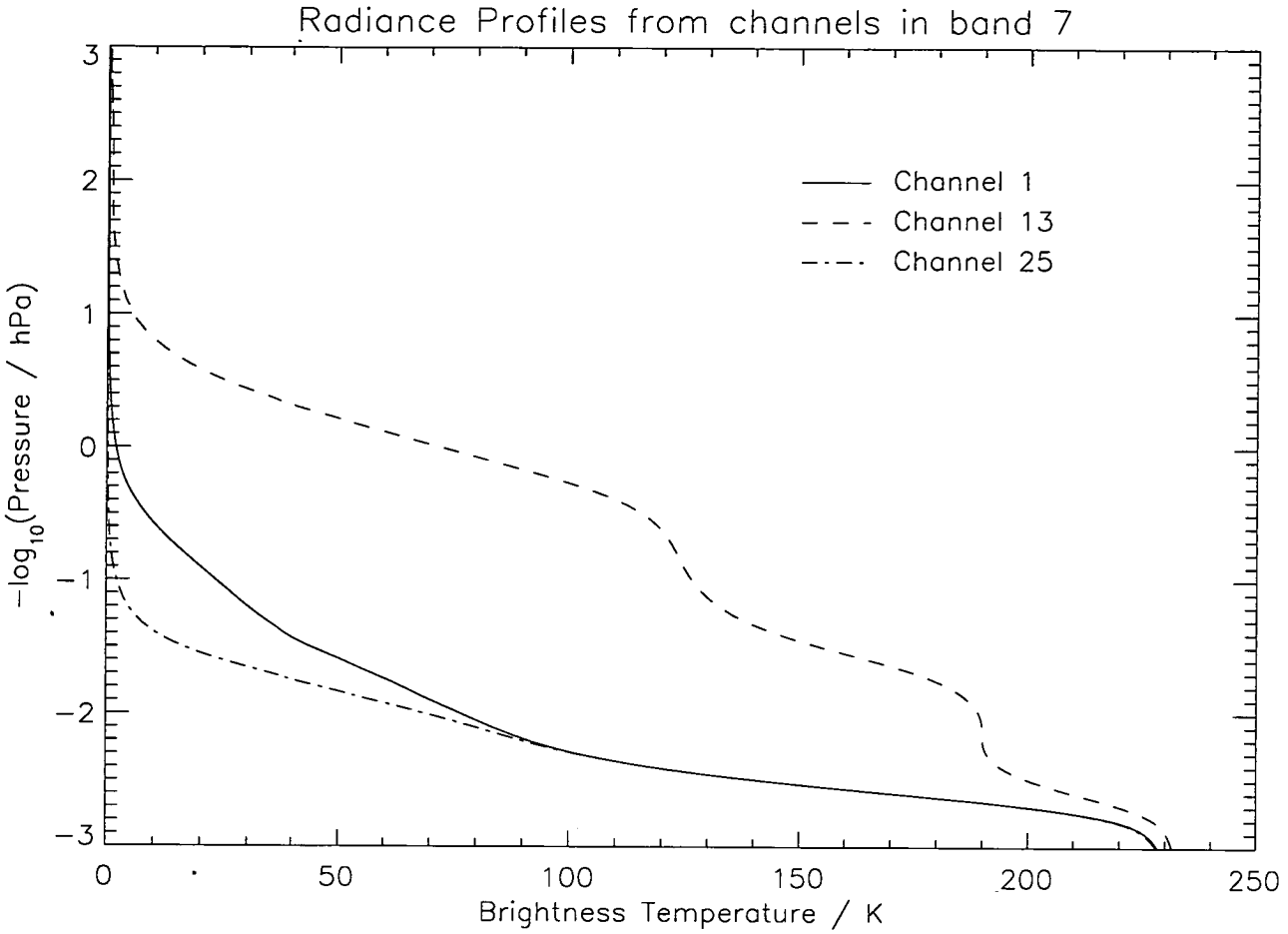

Figure 5.14: Example radiance profiles from channels 1, 13 and 25 of band 7 generated using temperature, ozone and water vapour. This band is highly non-linear as demonstrated by channel 13s profile.

order of $40 - 100$ K and so is necessary to include it. The effect on radiance of spectral lines against frequency for the lower and upper side bands of band 7 are shown in figures 5.15 and 5.16 respectively, in the same format as figure 5.9.

Due to time constraints, only two channels were modelled: channel 1 and channel 13. These were chosen as they represent the extremes of height that the band gathers information at. The instrument noise in these channels is $\sigma = 0.37$ K and $\sigma = 0.31$ K for channels 1 and 13 respectively. Three network configurations were tried.

First, a network whose inputs consisted of the tangent pressure, ozone and temperature profiles was trained. This was found to produce errors of around

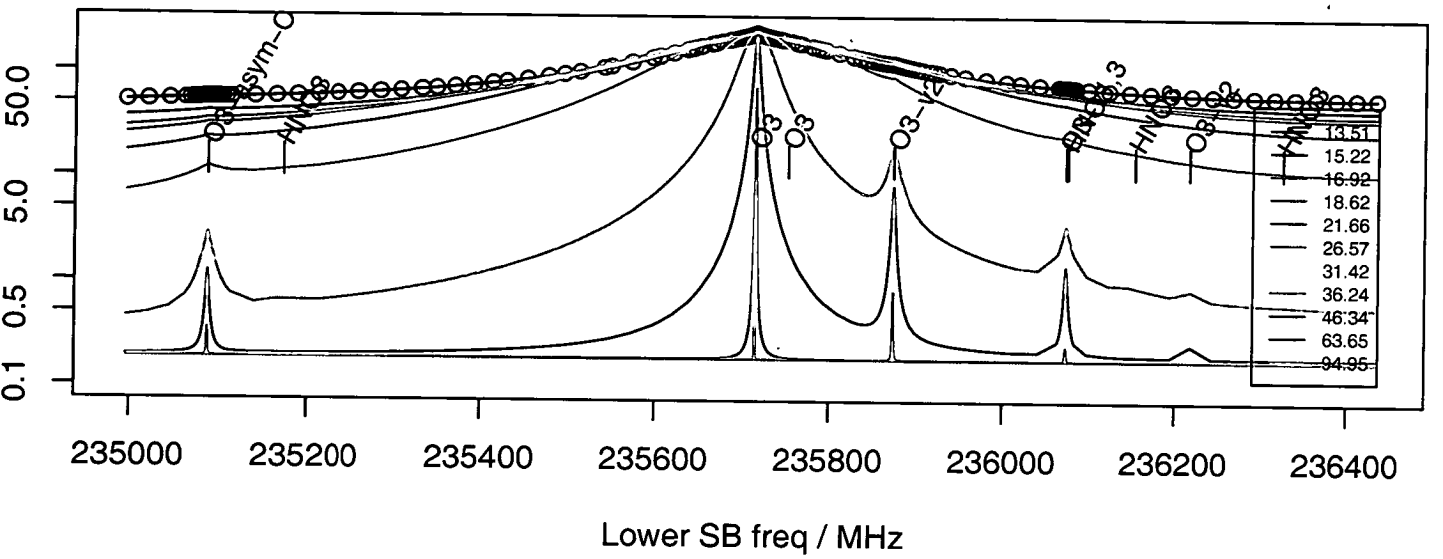

Figure 5.15: Frequency range for the lower side band of Band 7 showing the effect of spectral lines on radiances (similar to figure 5.9). Here oxygen, nitric acid, ozone and water vapour are used in the forward model.

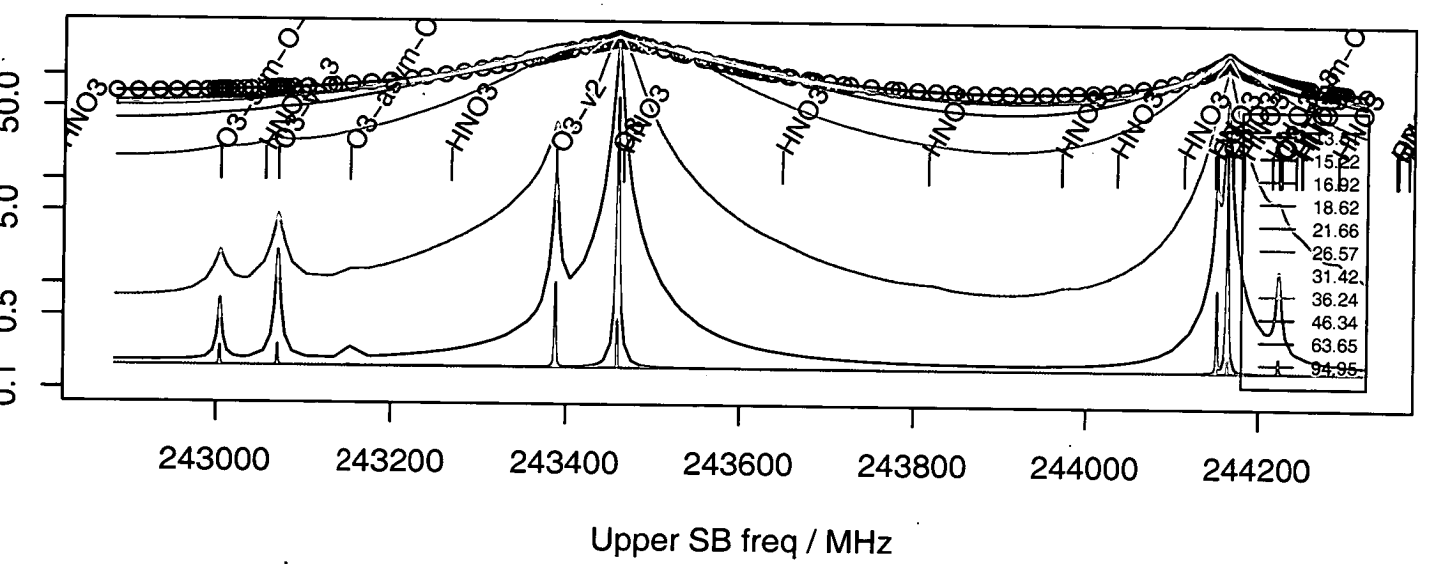

Figure 5.16: Frequency range for the upper side band of Band 7 showing the effect of spectral lines on radiances(similar to figure 5.9). Here oxygen, nitric acid, ozone and water vapour are used in the forward model.

$\sigma = 5$ K in both the modelled channels.

The second network configuration has tangent pressure, ozone, temperature and water vapour profiles as inputs. When using this network, the training error dropped to around $\sigma = 3$ K for both channels. While an improvement, this error is still unacceptably large.

The third network configuration removed the temperature profile from the inputs while keeping the tangent pressure, water vapour and ozone profiles. Training a network in this configuration improved the network error dramatically to produce a worst error of $\sigma = 1.45$ K for channel 1 and $\sigma = 0.75$ K for channel 13. Although still well above instrument noise levels, this error is localised around $z = -2.7$. Above $z = -2.2$, the error in both channels drops well below instrument noise to $\sigma = 0.2$ K for both channels. The results from both channels can be seen in figures 5.17 and 5.18 for channel 1 and 13 respectively.

A possible reason for this large error in the lower section of the profile can be seen in figure 5.19. The large increase in water vapour below $z = -2.2$ corresponds well with the height of the large error in the neural network. Above this level, there are only trace amounts of water vapour and the neural network is able to model the radiances well. Below $z = -2.2$, the water vapour level increases and the neural network error increases substantially.

As this problem only affects the tangent heights below $z = -2.2$ or $p \approx 160$ hPa, and below this height very little ozone is present in the atmosphere, the neural network can be used with these channels as part of an assimilation process for ozone. To do this, only those minor frames above $z = -2.2$ would be considered as part of the assimilation process. Several possible methods for dealing with water vapour are discussed briefly in chapter 6.

5.4 Discussion

This chapter has shown that it is possible to extend the neural network in several ways. It has been shown that the neural network can generate radiances from different channels and that the network can be extended to work in more realistic atmospheres where additional chemical species affect the radiances.

The initial part of the chapter looked at extending the neural network developed in chapters 3 - 4 for use in different channels. It was shown that most

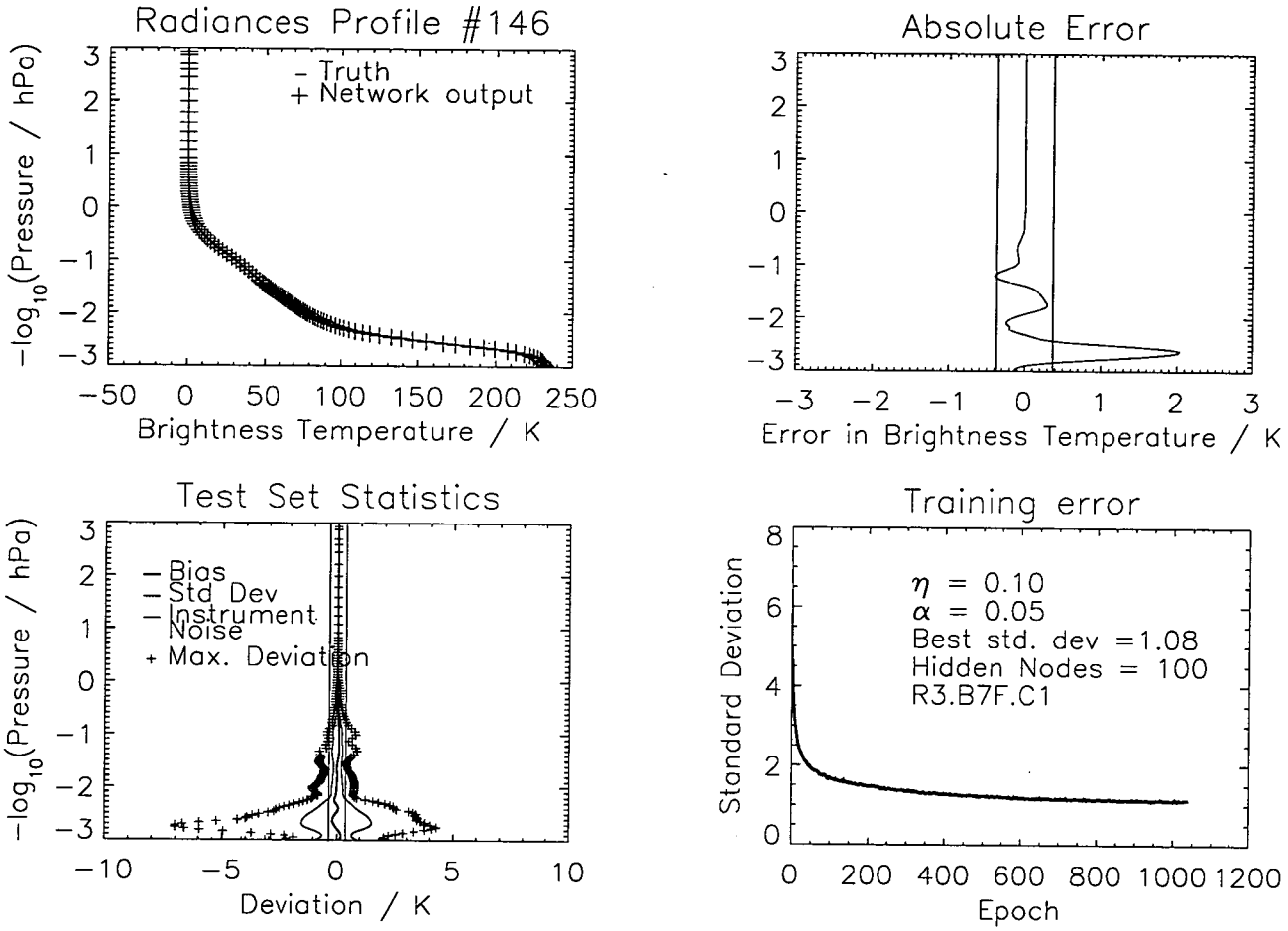

Figure 5.17: A training run for channel 13 of band 7 with ozone and water vapour as inputs (format is the described in section 3.5.1). Here, the worst standard deviation is around 1.45 K near $z = -2.7$. Above $z = -2.2$, the network is well trained with the worse error being around $\sigma = 0.25$ K.

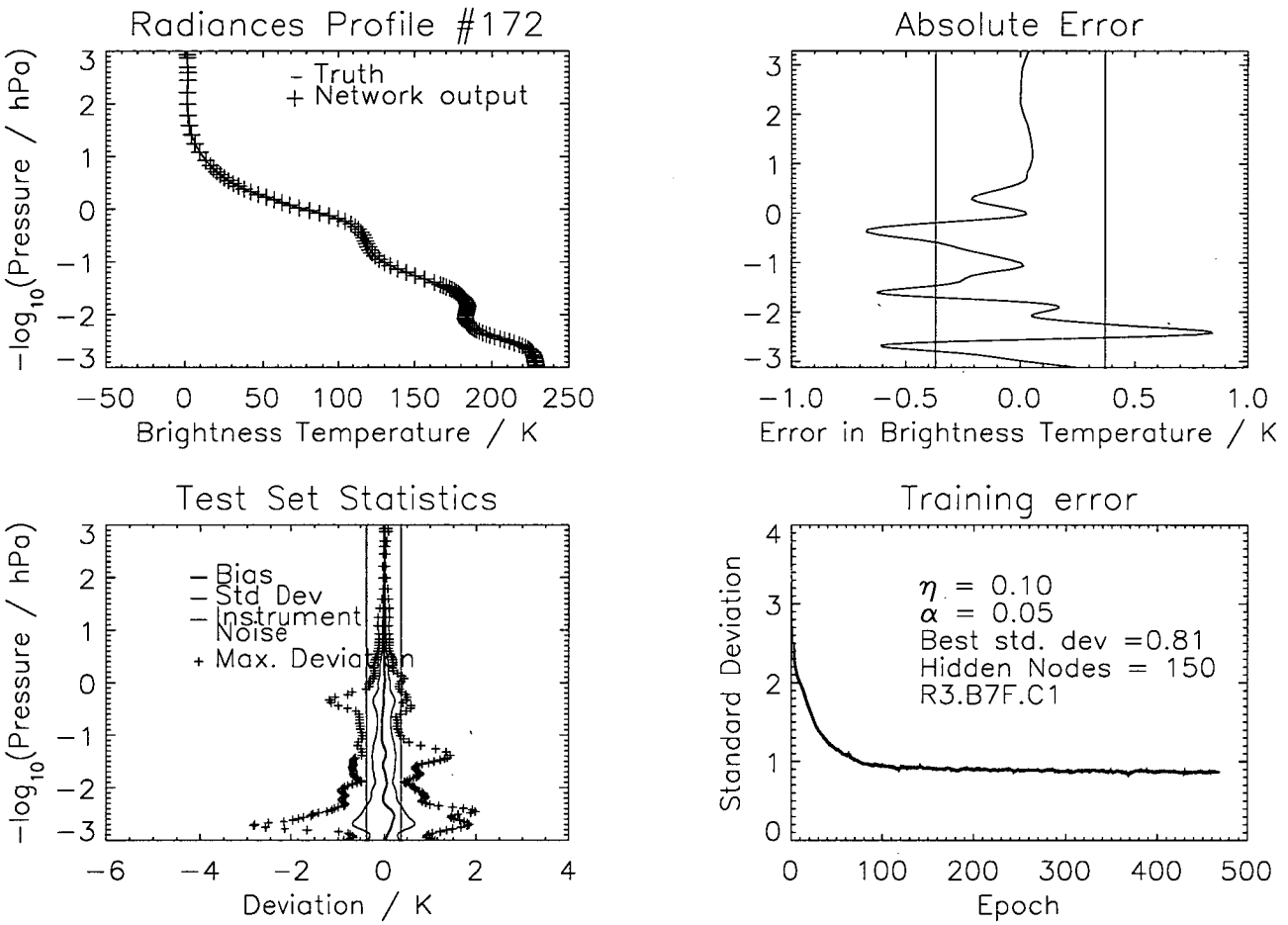

Figure 5.18: A training run for channel 13 of band 7 with ozone and water vapour as inputs (format is the described in section 3.5.1). Here, the worst standard deviation is around 0.75 K near $z = -2.7$. Above $z = -2.2$, the network is well trained with the worse error being around $\sigma = 0.2$ K.

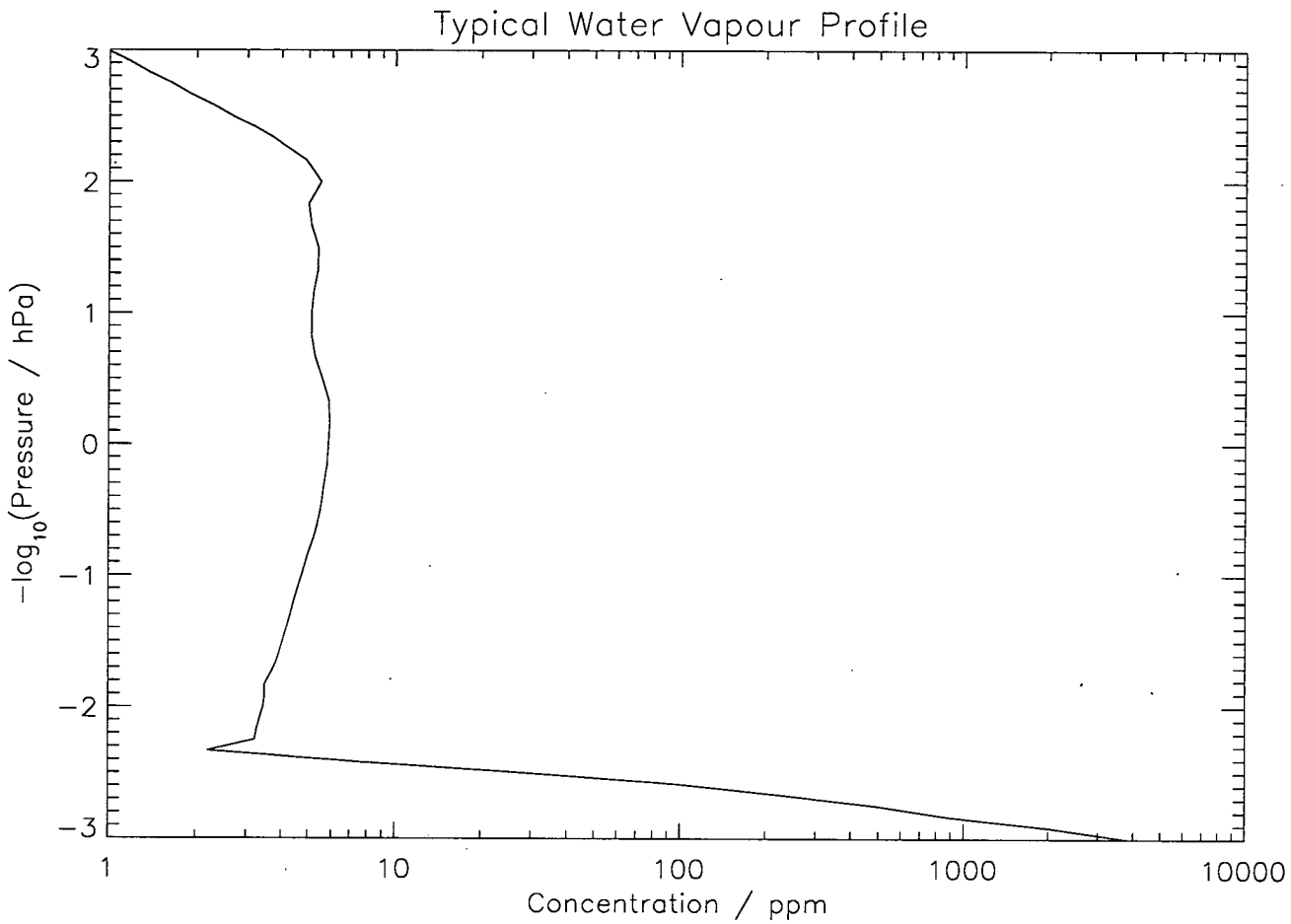

Figure 5.19: A typical water vapour profile used when running the forward model. Above $z = -2.2$, there is only a small amount of water vapour in the atmosphere which does not affect the produced radiances significantly. Below $z = -2.2$, the amount of water vapour increases significantly.

channels could be modelled within instrumental error levels. Several channels presented problems, but retraining these channels using an updated training set based on new instrument specifications resulted in training errors well below instrument noise levels.

The second part of the chapter looked at adding more chemical species to the neural network, resulting in a more realistic forward model. Initially, O3 and H2O were added to one channel in band 1, the channel most affected by these species. It was shown that the neural network could cope with these species, producing errors less than the instrument noise level.

Several channels from another band were modelled using a neural network. This band, band 7, is centered on an ozone line and has a highly non-linear response. It was found that the radiances for these channels could be well modelled above 160 hPa. Below this, the effects of water vapour dominate the radiances and the neural network is unable to cope, producing errors of around $\sigma = 1$ K. The failure of the neural network may be due to the large range of values encountered in water vapour, which varies proportionally much more at one height than other species examined. This may mean the training data is much less representative or more sparsely spaced, resulting in the neural network being unable to learn the data correctly. One solution to this would be to increase the training dataset size, allowing more coverage of the expected range of water vapour values.

Due to time constraints, the effects of water vapour on band 7 radiances were not thoroughly explored. One approach that was examined briefly was to use a neural network to calculate only the bottom 30 minor frames (to around 200 hPa). This reduced the error for those minor frames in channel 1 but the resulting network still had an error much larger than the instrument noise.

The work on incorporating chemical species in this chapter has focused on ozone as this is normally part of the assimilation models state vector. It should be possible to deal with other species in a similar way. Problems will arise when adding a species that are not part of the assimilation process's state vector to the forward model as the species profile must be specified externally. It should be noted that, as the neural network is a straight replacement for a traditional forward model, these problems must also be faced when using a traditional forward model.

This chapter has extended the neural-network-based forward model to sim-

ulate a more realistic atmosphere. It has shown that the network can work in different channels and that it is possible to handle species information in some cases. Chapter 6 looks the final problem that must be overcome for a neural-network-based forward model to be considered - providing a Jacobian for the neural network.

Chapter 6

The Adjoint Model

6.1 Introduction

Previously, it has been shown that a neural network can replicate a forward model for the EOS-MLS well. However, in order to integrate this neural network into an assimilation scheme, an adjoint model is also required.

A 4D-VAR assimilation scheme is a three-step process. First, the model fields are used to generate expected instrumental radiances using a forward model within a time window. These radiances are then compared to the real instrumental radiances at the same location and the error established. In the final step, the model fields are updated for this time window using the adjoint model.

In this chapter we discuss what is involved in the adjoint model and how this can be achieved using a neural network forward model. It shows that an adjoint model can be constructed using a neural network that may suitable for use in an assimilation scheme.

6.2 The Adjoint Model

Chapter 2 discusses the assimilation process of a 4D-VAR system in detail. The framework is reiterated here with an emphasis on the adjoint model.

In a 4D-VAR assimilation scheme, the system evolves according to equations 6.1 and 6.2, where $\vec{x}_k$ is the state of the system at time-step k (where an observation is made), $\vec{u}_k$ are the inputs to the dynamical model at time-step k (e.g. ground albedo values) and f_k is a nonlinear function describing the evolution

of the system, through the dynamical equations, between successive observation times, $k = 0, \ldots, N-1$. The observations are related to the system states by way of the radiative transfer equation (6.2), which has an error term, δ_k which is assumed to be unbiased, uncorrelated in time and Gaussian with covariance matrix R_k.

$$\vec{x}_{k+1} = f_k(\vec{x}_k, \vec{u}_k), \qquad k = 0, \ldots, N-1 \tag{6.1}$$

$$\vec{y}_k = h_k(\vec{x}_k) + \vec{\delta}_k \tag{6.2}$$

The assimilation is achieved by minimising the cost function, J, given by equation 6.3 with respect to x_0, where $\vec{x}_0^b$ is the initial background state with error covariance B_0, which is assumed to be known.

$$J = \frac{1}{2}(\vec{x}_0 - \vec{x}_0^b)^T B_0^{-1}(\vec{x}_0 - \vec{x}_0^b) + \frac{1}{2}\sum_{k=0}^{N-1}(h_k(\vec{x}_k) - \vec{y}_k)^T R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k) \tag{6.3}$$

This problem can then be solved iteratively using a gradient descent method. The cost function is first split into two parts (equation 6.4) where J_0 and J_i are given by equations 6.5 and 6.6 respectively.

$$J = \frac{1}{2}J_0 + \frac{1}{2}\sum_{k=0}^{N-1} J_i \tag{6.4}$$

$$J_0 = \left(\vec{x}_0 - \vec{x}_0^b\right)^T B_0^{-1}\left(\vec{x}_0 - \vec{x}_0^b\right) \tag{6.5}$$

$$J_i = \left(h_k\left(\vec{x}_k\right) - \vec{y}_k\right)^T R_k^{-1}\left(h_k\left(\vec{x}_k\right) - \vec{y}_k\right) \tag{6.6}$$

Two assumptions are then made. The first assumption is that the states of the model, $\vec{x}_k$, can be expressed in terms of the initial state, $\vec{x}_0$, as $\vec{x}_k = f_k(f_{k-1}(\ldots f_0(\vec{x}_0, \vec{u}_0)))$. The second assumption is that both f_k and h_k can be linearised around the current trajectory, using equations 6.7 and 6.8, where F_k and H_k are the Jacobians of f_k and h_k with respect to $\vec{x}_k$.

$$\vec{x}_{k+1} = f_k(\vec{x}_k, \vec{u}_k) + F_k\vec{\epsilon}_k \tag{6.7}$$

$$h_k(\vec{x}_k) - \vec{y}_k \approx F_k H_k \vec{\epsilon}_{k-1} - \vec{y}_k \tag{6.8}$$

Using these relations, along with the constraints given by equation 6.1, the gradient of the cost function can be derived as in equations 6.9 - 6.12, where $d_k = R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k)$ is called the departure of the observation and $\nabla_{\vec{x}_0}$ is the derivative with respect to $\vec{x}_0$.

$$\nabla_{\vec{x}_0} J = \quad \nabla_{\vec{x}_0} J_0 + \sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i \tag{6.9}$$

$$\nabla_{\vec{x}_0} J = \quad B_0^{-1}(\vec{x}_0 - \vec{x}_0^b) + \sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i \tag{6.10}$$

$$\sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i = \quad \sum_{k=0}^{N-1} F_1^T F_2^T \ldots F_k^T H_k^T d_k \tag{6.11}$$

$$\sum_{k=0}^{N-1} \nabla_{\vec{x}_0} J_i = H_0^T d_0 \quad + \quad F_1^T (H_1^T d_1 + F_2^T (H_2^T d_2 + \ldots F_{N-1}^T H_{N-1}^T d_{N-1})) \tag{6.12}$$

Defining λ_k by equation 6.14, the gradient of the cost function can be rewritten as in equation 6.15. λ_k are the adjoint variables, which measure the sensitivity of the gradient to changes in the k^{th} measurement.

$$\lambda_N = 0 \tag{6.13}$$

$$\lambda_k = F_k^T(\vec{x}_k)\lambda_{k+1} \quad - H_k^T R_k^{-1}(h_k(\vec{x}_k) - \vec{y}_k) \tag{6.14}$$

$$\nabla_{\vec{x}_0} J = B_0^{-1}\left(\vec{x}_0 - \vec{x}_0^b\right) - \lambda_0 \tag{6.15}$$

It is assumed that F_k and R_k are known in equation 6.14. H_k is the Jacobean of the instrument forward model, h_k. This H_k must be calculated and supplied by the forward model for each forward model calculation. This chapter discusses how H_k may be calculated for the neural-network forward model.

6.3 Calculating the Jacobian

In traditional forward models, the Jacobian of the instrument's forward model is generally found by using an automatic differentiation routine (e.g. Giering (1999)). This process takes in a (FORTRAN) routine and produces a corresponding routine for calculating the derivative of this function.

With a neural network, automatic differentiation would be slow and error prone. During the running of the neural network, derivatives of the activation functions for each node are calculated for training (see section 2.5.4 in chapter 2). An automatic differentiation scheme would recalculate them numerically, resulting in a large slowdown and may introduce numerical errors due to non-analytical differentiation.

Instead, the neural network may be differentiated by hand and then implemented in code. The general equation for output q of a neural network with one hidden layer of m nodes is given by equation 6.16 (derived from Krasnapolsky (1997)) where y_q is the output value of the node q, b_q and a_q are normalisation constants, ϕ is the activation function of the output node. ω_{qj} is the weight from hidden node j to the output node q, γ is the activation function of the hidden node, Ω_{ji} is the weight from the input node i to the hidden node j and $I(i)$ is the input value of node i. $\min^* I(i)$ and $\max^* I(i)$ are the (constant) minimum and maximum values of the training set for the input node i including scaling factors, as defined in section 3.3 of chapter 3. B_j and β_q are the network biases for the hidden node j and the output node q respectively.

$$y_q = b_q + a_q \phi \{ \sum_{j=1}^{m} \omega_{qj} \left[\gamma \left(\sum_{i=1}^{n} \Omega_{ji} \left(\frac{I(i) - \min^* I(i)}{\max^* I(i) - \min^* I(i)} \right) + B_j \right) \right] + \beta_q \} \quad (6.16)$$

In this case, both ϕ and γ are the sigmoid function 6.17, which has a derivative of 6.18. When written this way, it is possible to differentiate the equation for y_q analytically using the chain rule. This results in equation 6.19 for the derivative of y_q with respect to input I_a where z_j is the output of the hidden node j and V_q is the (unnormalised) output of the output node q.

$$ac(\sigma) = \frac{1}{1 + exp(-\sigma)} \quad (6.17)$$

$$\frac{d}{d\sigma}ac(\sigma) = \sigma(1-\sigma) \tag{6.18}$$

$$\frac{\partial y_q}{\partial I_a} = a_q\left(V_q\left(1-V_q\right)\right)\sum_{j=1}^{m}\omega_{qj}\Omega_{ja}\left(z_j\left(1-z_j\right)\right)\frac{1}{\max^* I(a) - \min^* I(a)} \tag{6.19}$$

In equation 6.19, the term a_q is a normalisation constant used to convert (with b_q in equation 6.16) the output from the range $\{0, 1\}$ to the actual output. The actual value of this is defined by equation 6.20 where $\max^* O(q)$ and $\min^* O(q)$ are the scaled maximum and minimum values in the training set for output q, discussed in section 3.3 of chapter 3.

$$a_q = \max^* O(q) - \min^* O(q) \tag{6.20}$$

All the values in equation 6.19 are easily found in the neural network program and have already been calculated on the forward pass. Using the pre-generated values, it is easy to calculate the Jacobian of the neural network at very little cost in time. The issue faced here is whether this neural-network generated Jacobian is accurate enough, compared to the true Jacobian, to be used in an assimilation scheme.

6.4 Results

To create the Jacobian of the network, the formula 6.19 must be applied to each output for each input. In the neural network, this creates an array of $[193, 120]$ numbers as there are 193 inputs to the neural network (73 temperature inputs and 120 tangent pressures) and 120 outputs (radiances).

Figure 6.1 shows an example Jacobian for temperature generated using automatic differentiation of the true forward model. This Jacobian is generated from channel 1 of band 1. The main feature of this is the large red bulge below $z_y = -1.7$ and the large negative (blue) values above this, around $z_y = -1.2$. This means that for all tangent $z < -1.7$, the radiances contain information about the temperature near $z = -1.7$ while around $z = -1.2$, the radiances are inversely proportional to the temperature (the influence is negative). This

negative influence arises principally from the temperature dependence of the absorption coefficient.

Using the neural network, the corresponding Jacobian with respect to temperature can be seen in figure 6.2. Here, the negative section, below $z_y = -2.7$, is visible though the values are smaller than the true Jacobian. The negative area, around $z_y = -1.2$ in the true Jacobian, is less well defined in the neural network Jacobian and is more horizontal. This shows that the network Jacobian is unacceptable for use in this case.

Retraining the network used, the validation error was reduced from 0.13 with 45 hidden nodes to 0.11 with 55 hidden nodes. This was achieved by updating the network to use the new training data described in section 5.2.3 in chapter 5. Previously, the network had 120 outputs and 193 inputs. When using the new instrument specifications, the network was increased to 125 outputs and 222 inputs. The Jacobian in this case was found to be still too noisy.

Up until now, the network had been trained using sigmoid transfer functions. Chapter 3 discussed the use of hyperbolic tangent transfer functions, but their use was rejected as they tended to send the error towards infinity, if not used with care, moreover they provided no significant advantage over sigmoid functions. Here, they are again considered. The previous network-generated Jacobians may have been inaccurate due to the network finding local minima in weight-space instead of the global minimum. By using a hyperbolic tangent transfer function, this problem may be avoided as the previous behaviour of sending the error towards infinity might allow it to pass these local minima and find the global minimum.

When the network is trained using hyperbolic transfer functions, together with the new training data, the validation error was again around 0.11. The derivative of the network, equation 6.19, can be adapted when using hyperbolic tangent transfer functions to equation 6.21, where the symbols are as described previously (see section 6.3).

$$\frac{\partial y_q}{\partial I_a} = a_q \left(\left(1 - V_q^2\right)\right) \sum_{j=1}^{m} \omega_{qj} \Omega_{ja} \left(\left(1 - z_j^2\right)\right) \frac{1}{\max^* I(a) - \min^* I(a)} \tag{6.21}$$

The network Jacobian in this case is shown in figure 6.3. As can be seen,

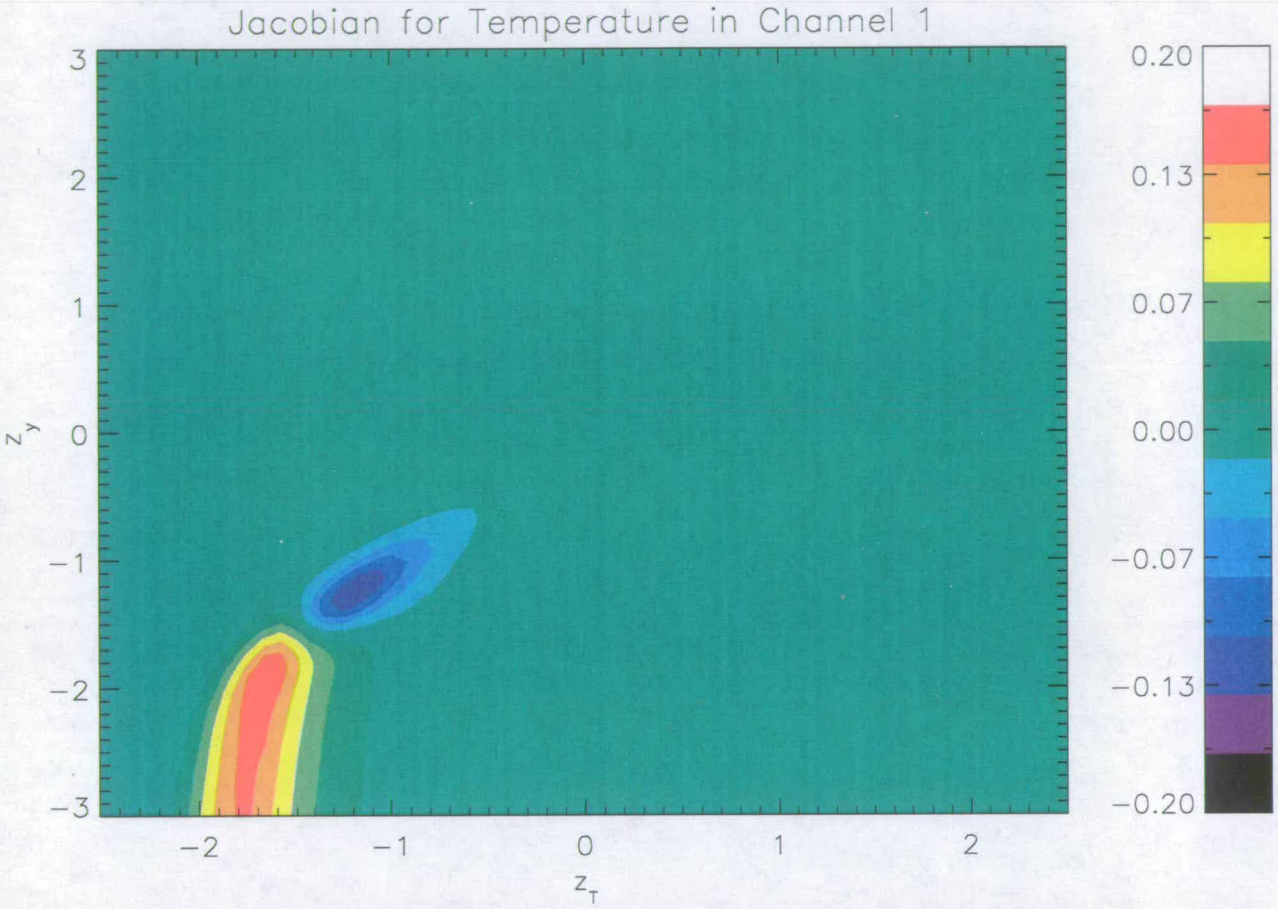

Figure 6.1: The Jacobian for temperature, for channel 1 of band 1 generated using automatic differentiation of the true forward model. Each input to the forward model will have 125 entries in the Jacobian (one value for each output), and is represented by a vertical slice in the diagram at the corresponding height (in log-pressure space), z_T. z_y corresponds to the tangent pressure of the measured radiance and the Jacobian value at that point is represented by a colour, indicated by the scale on the right. This shows that for $z < -1.7$, the radiances in this channel contain information about the temperature near z=-1.7 (The channel is blacked out)

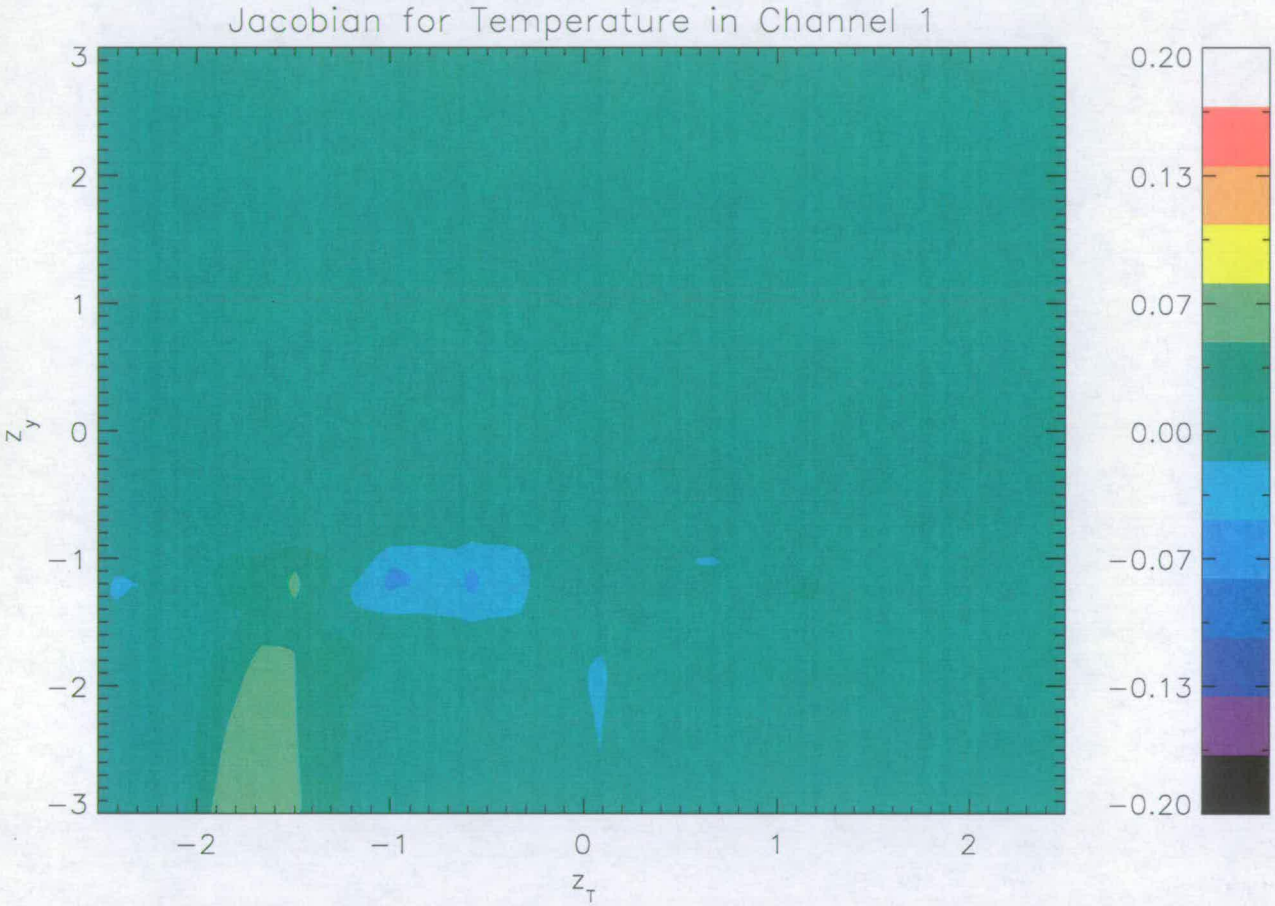

Figure 6.2: The Jacobian for temperature, for channel 1 of band 1, generated using differentiation of the neural network. Compared to figure 6.1, it can be seen that the main features are similar but there are a lot of extra features.

this is much closer to the true Jacobian. There are still several unaccounted for discrepancies (e.g. around $z = [1.0, -1.3]$) however these are have values around 0.02. The difference from the truth is shown in figure 6.4. The largest error is around 0.05 and occurs near the large negative section around $z = [-1, -1]$. This error means that a $1K$ change in temperature at that height will result in error of $0.05K$ in the radiance.

Garand et al. (2001) propose a "goodness" measure for Jacobians of a nadir sounding forward model. This "goodness", M, can be defined as equation 6.22, where $J_{M,i}$ are the elements of the proposed Jacobian and $J_{R,i}$ are the elements of the true Jacobian. The summation is over all elements in the Jacobian.

$$M = 100\sqrt{\frac{\sum_{i=1}^{N}\left(J_{M,i} - J_{R,i}\right)^2}{\sum_{i=1}^{N} J_{R,i}^2}} \tag{6.22}$$

It is suggested that values of $M < 5$ indicate an excellent fit, $5 < M < 15$ are a good fit, and generally suitable for use in NWP applications. Values of $15 < M < 25$ are fair to marginal and $M > 25$ indicate a serious problem. This measure only gives an indication of whether the Jacobian is suitable. Examination of Jacobians within the assimilation process environment is needed to ensure the Jacobian is suitable.

This measure can be adapted to the limb sounding case by considering each minor frame as a separate measurement. In the case of the EOS-MLS, this produces a set of 125 M-values, which can be plotted. For the Jacobian in this case, the results can be found in figure 6.5. In this case, below $z_y < 1.5$, the M value is around 10. Above this, the Jacobian is effectively zero and the M value tends towards infinity. This suggests that the Jacobian may good enough to use in an assimilation scheme.

The largest errors in the network Jacobian occurs in a small selection of minor frames near $z_y = -1.3$. To investigate whether the model could be made more accurate in this area, a reduced neural network was constructed. This network consisted of one output, the radiance for one minor frame near this height. The inputs consisted of 1 tangent pressure for the radiance and 36 temperature inputs from $z < 0.0$. It was found that using 5 hidden nodes produced optimal results, with a testing set error of $\sigma = 0.10K$. The Jacobian generated from this can be seen in figure 6.6 and is equivalent of taking a horizontal slice through figure 6.3

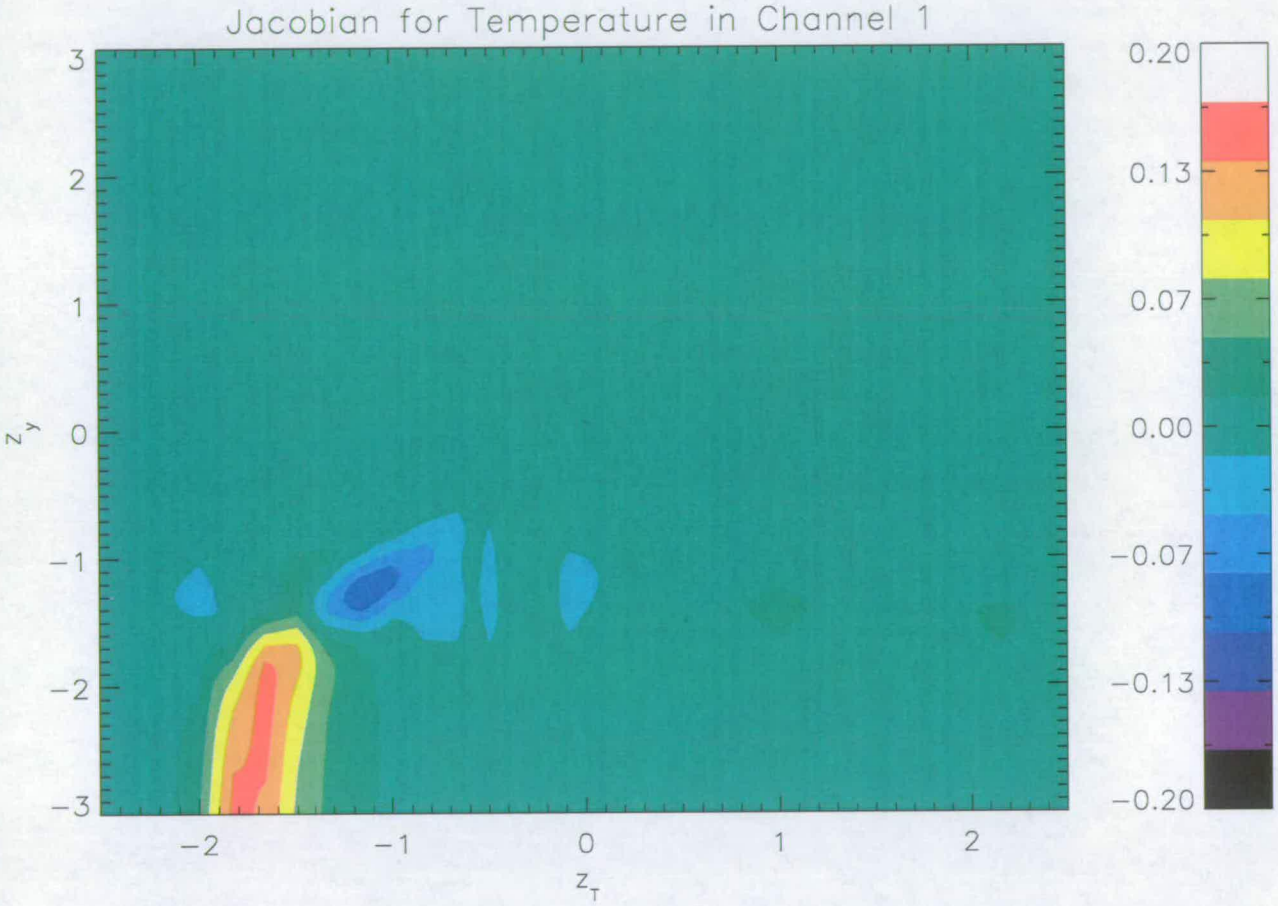

Figure 6.3: The Jacobian for temperature, for channel 1 of band 1, generated using a neural network with hyperbolic tangent transfer functions. Compared to figure 6.2, it is now much cleaner and the main features are much more pronounced. There are still small errors outside the main feature but these have a size of around 0.02.

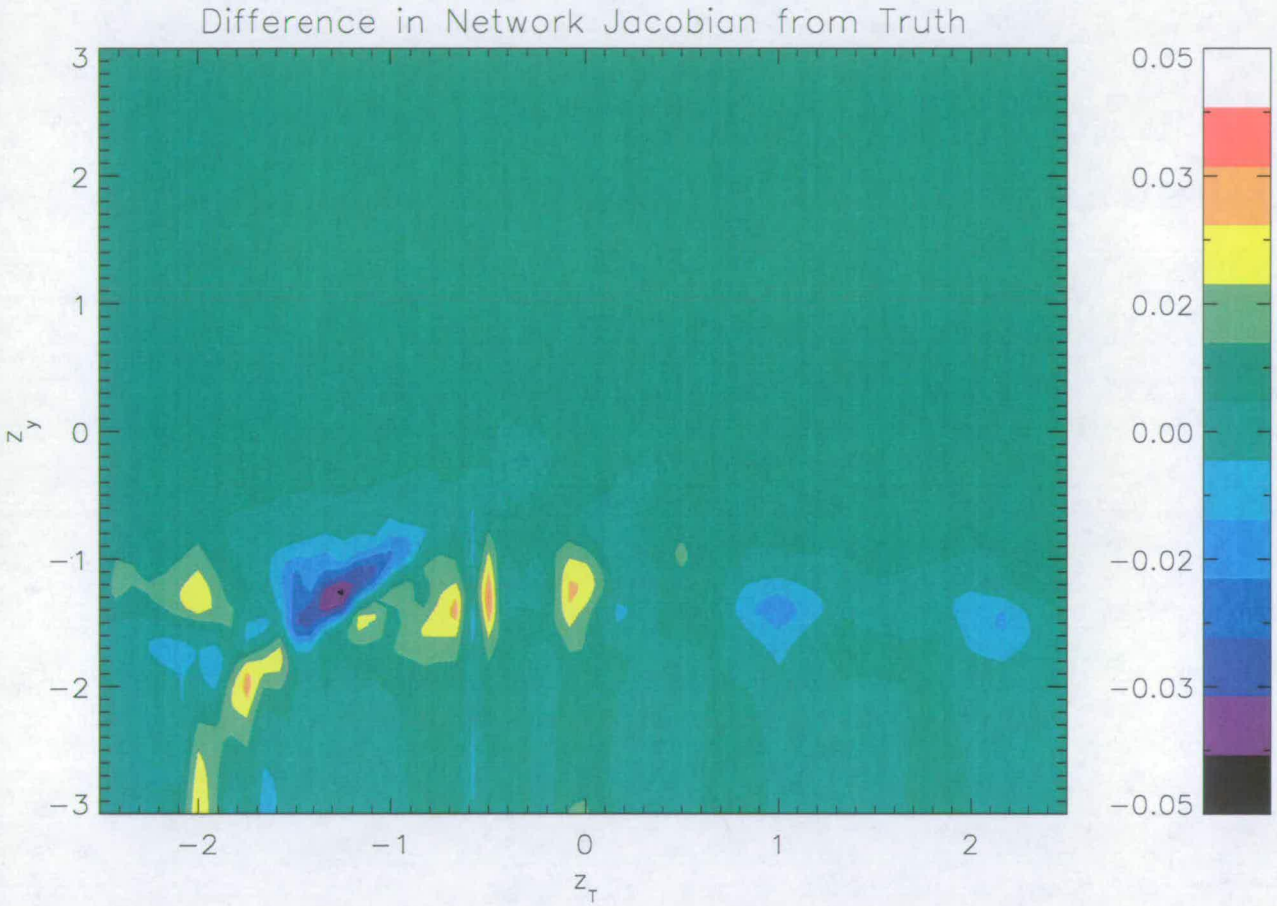

Figure 6.4: The difference between the true Jacobian (figure 6.1) and the network-generated Jacobian (figure 6.3). The largest error is approximately 0.05 in value which corresponds to around $2K$ error in temperature.

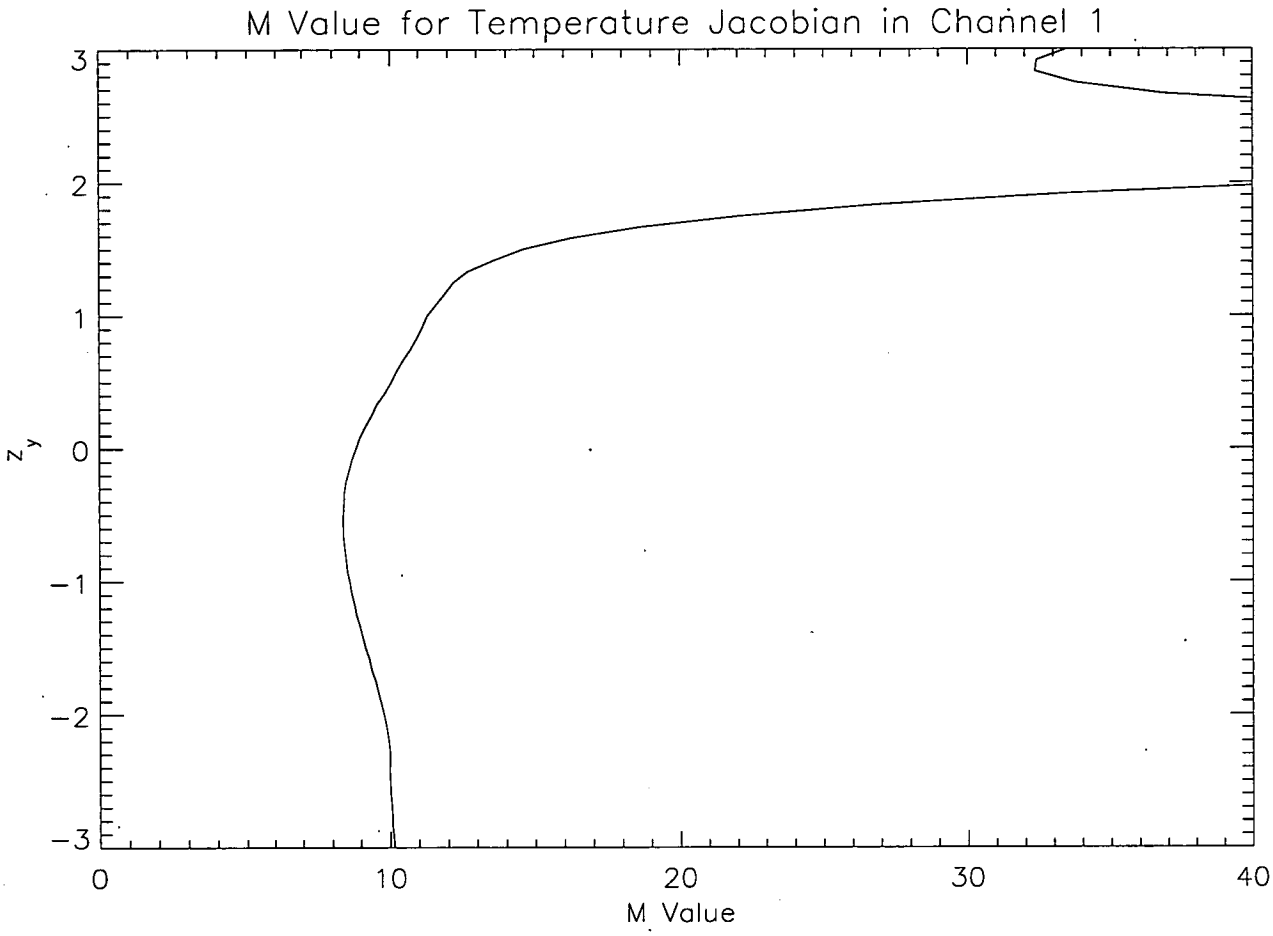

Figure 6.5: The M-value plot for the network-generated Jacobian (figure 6.3). This gives an approximation of how good the generated Jacobian is. Values of $M < 15$ are considered good. Here, for $z_y < 1.5$, the M values are approximately 10, suggesting the Jacobian is suitable for use in an assimilation scheme.

at $z_y = -1.3$. The error in this Jacobian is now less than 0.03, which shows that it is possible to further improve the Jacobian produced by the neural network.

Other channels within band 1 have similar results. Figures 6.7, 6.8 and 6.9 show the true Jacobian, the network Jacobian for temperature inputs and the difference from truth for channel 8. Here, the network has been retrained as above using hyperbolic tangent activation functions. The network validation error is approximately $\sigma = 0.1$ and the worst error in the Jacobian is around 0.06. As before, the sections outside the main negative section have values of less that 0.03. Here, the error in the negative section of the K matrix is much more pronounced. The M-values of this can be seen in figure 6.10. Here, the M values are approximately 16, which higher than channel 1 and may pose larger problems when integrating with an assimilation model.

This section has looked at the Jacobian of the neural network with respect to temperature. It has shown that a Jacobian for the neural network can be constructed analytically and that the resulting Jacobian may be sufficiently accurate for use in an assimilation scheme, though this would need further testing within the assimilation scheme environment. The analytical differentiation produces the same derivative as perturbation of temperatures, but is substantially quicker as all the intermediate variables are already available.

6.5 Tangent Pressure Jacobian

The role of tangent pressures in the forward model was previously discussed in chapter 4. It was shown that it is possible to retrieve the tangent pressures using a neural network outside the forward model. As previously stated, tangent pressures are not part of the assimilation state vector for technical reasons. However, the tangent pressures are still inputs into the forward model. It is therefore interesting to look at their derivatives.

In the true forward model, the section of the Jacobian related to the tangent pressure is highly sparse as the tangent pressure for one scan position has no effect on the radiances at any other scan positions. The non-zero elements are plotted in figure 6.11.

The network output is plotted in figure 6.12. In this case, it can be seen that the Jacobian is definitely not sparse but has a structure similar to the temperature

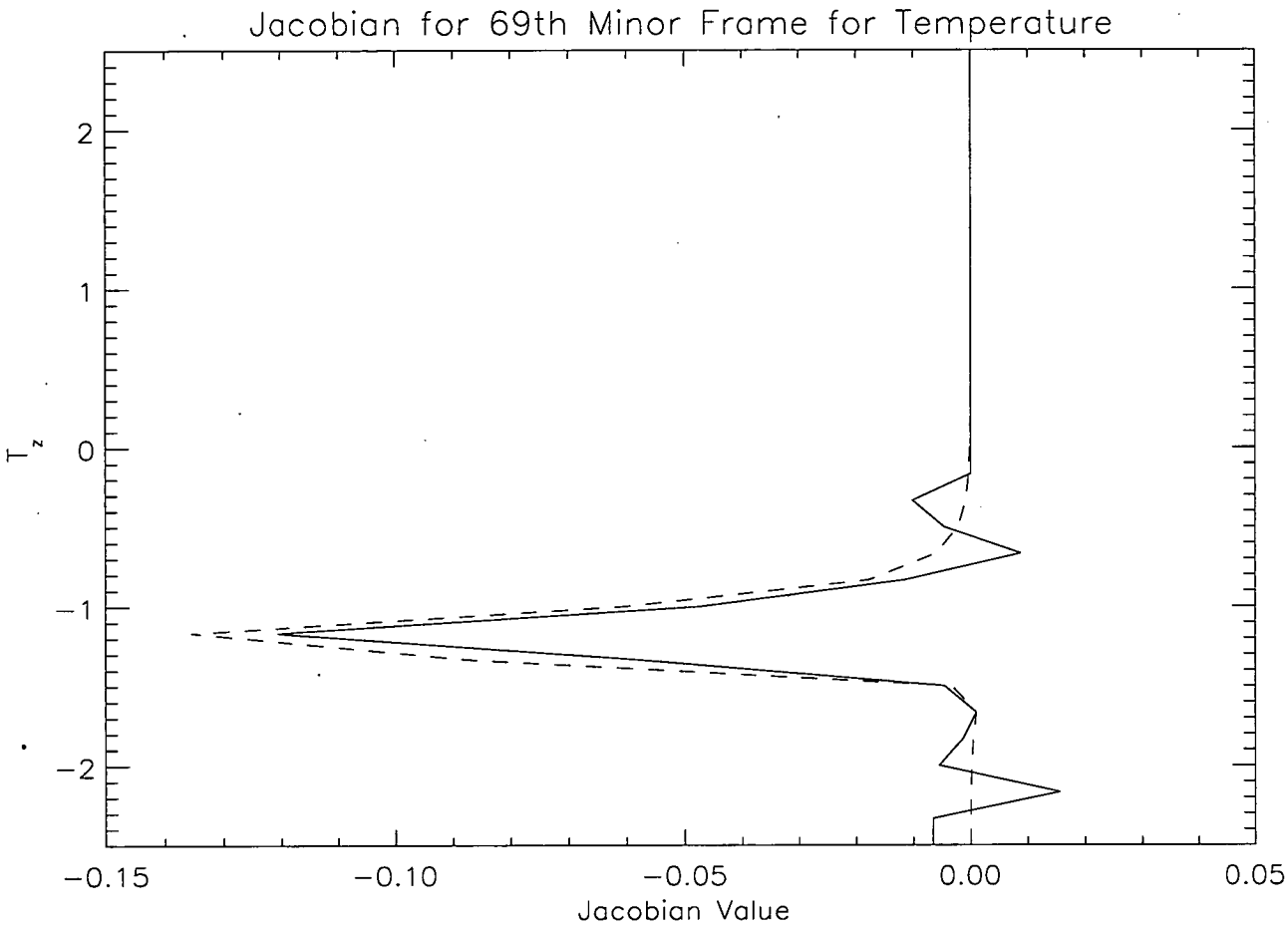

Figure 6.6: The Jacobian elements for temperature for one minor frame from the truth (dashed line) and a reduced neural network (solid line). The reduced neural network inputs consists of 36 temperatures (levels below $z = 0.0$) and the tangent pressure for the minor frame. The output is a single radiance for that level. Here, the error in the neural network Jacobian is always less than 0.03.

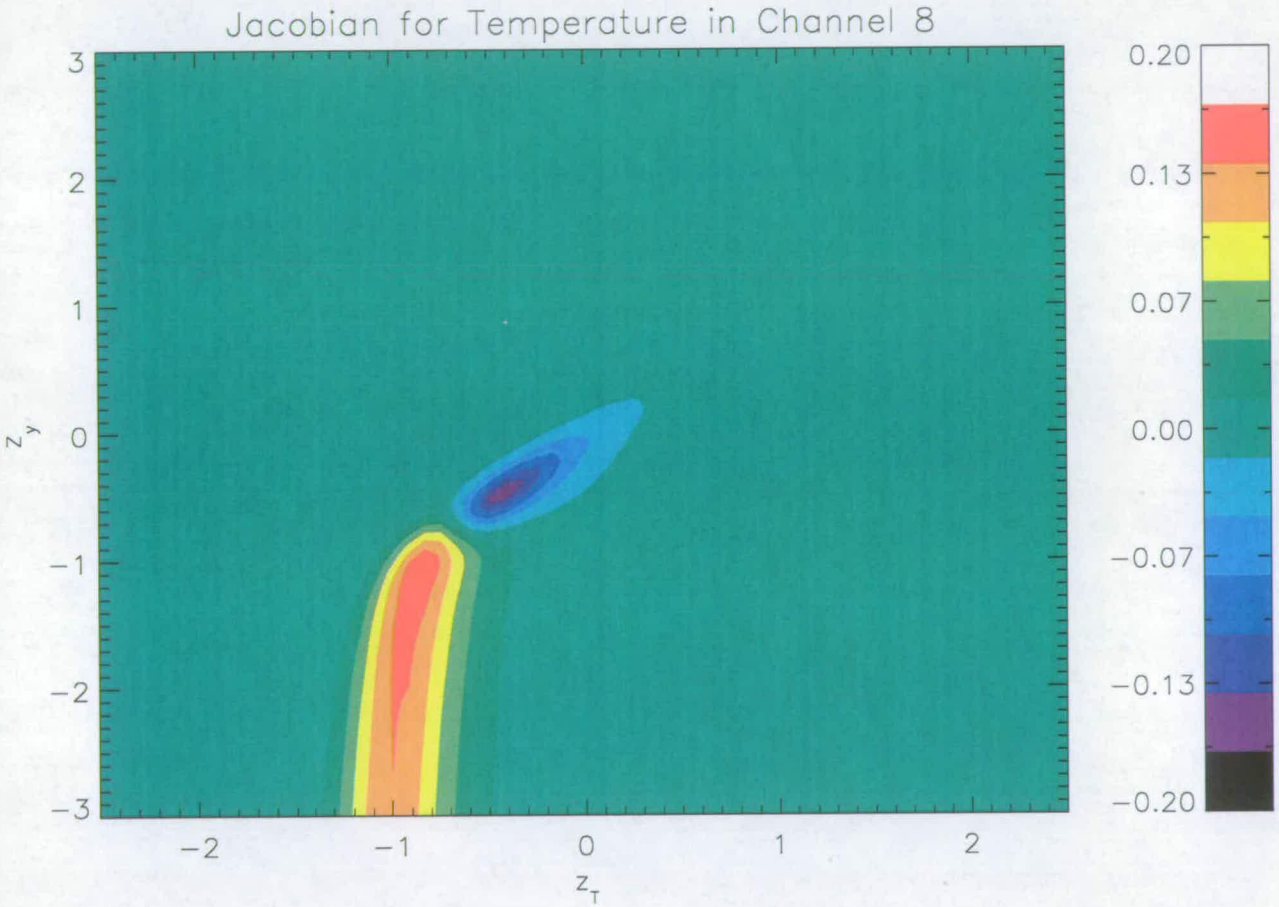

Figure 6.7: The Jacobian for the true forward model, generated using automatic differentiation for channel 8 of band 1.

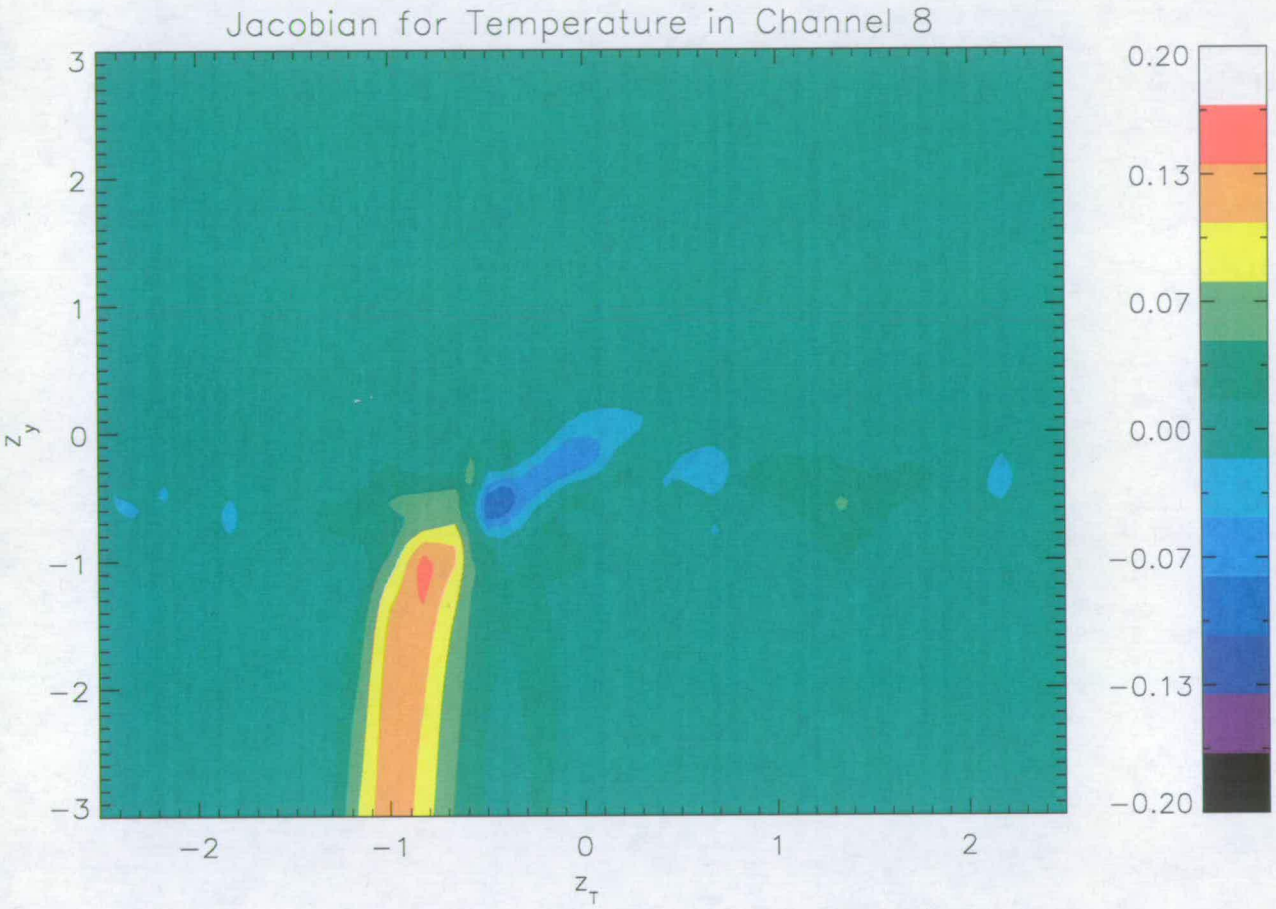

Figure 6.8: The Jacobian for the network for channel 8 of band 1. Although not as clean as the channel 1 Jacobian (figure 6.3), the large errors occur at only a small number of minor frames (see figure 6.9) which suggests a similar problem to the Jacobian from channel 1.

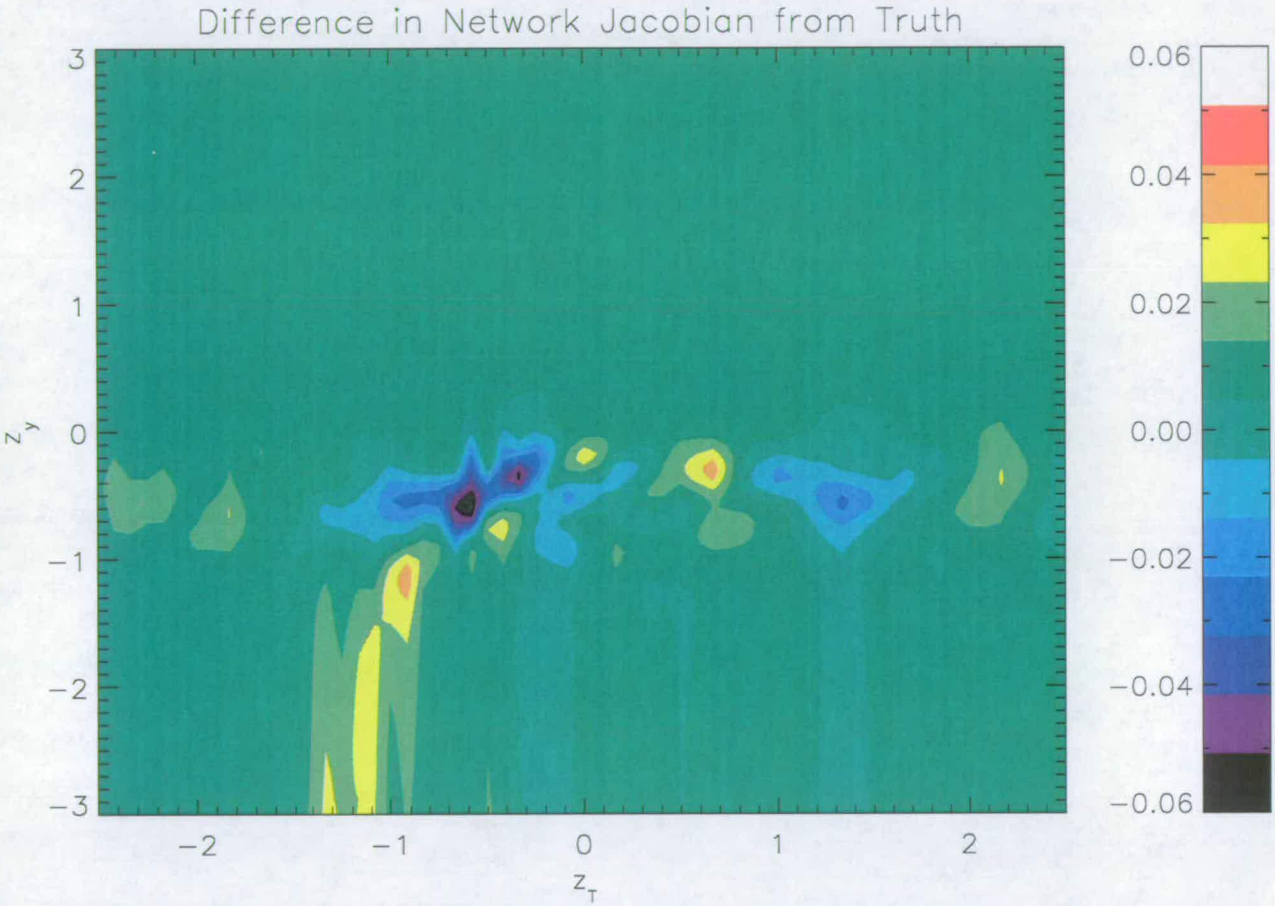

Figure 6.9: The difference between the network generated and the true Jacobians for channel 8. The largest error is around 0.06. Away from the large errors, the errors are less than 0.03.

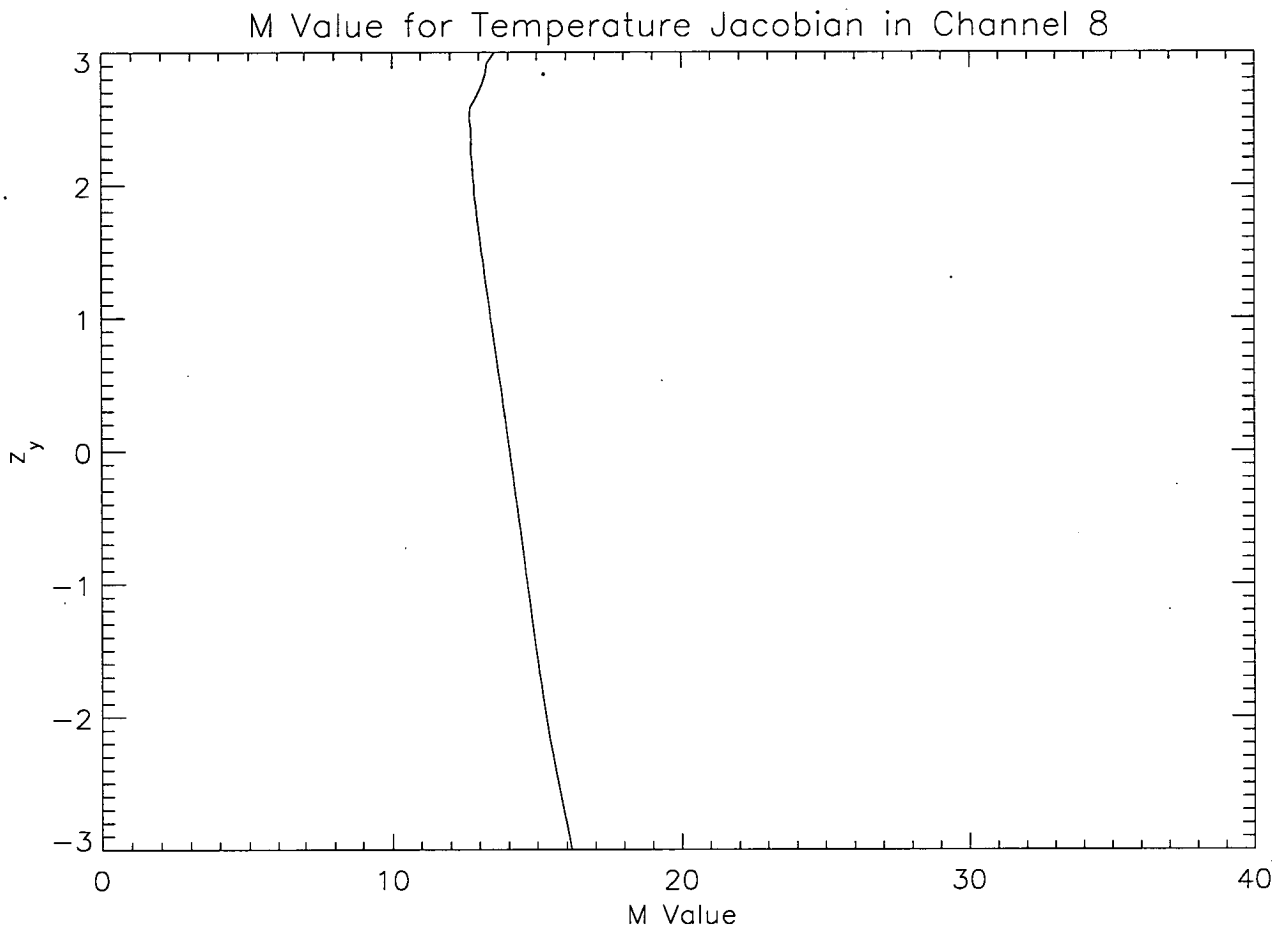

Figure 6.10: The M values for the network generated Jacobian for channel 8. Here, the M values are around 16. For comparison, channel 1 produced M values around 10.

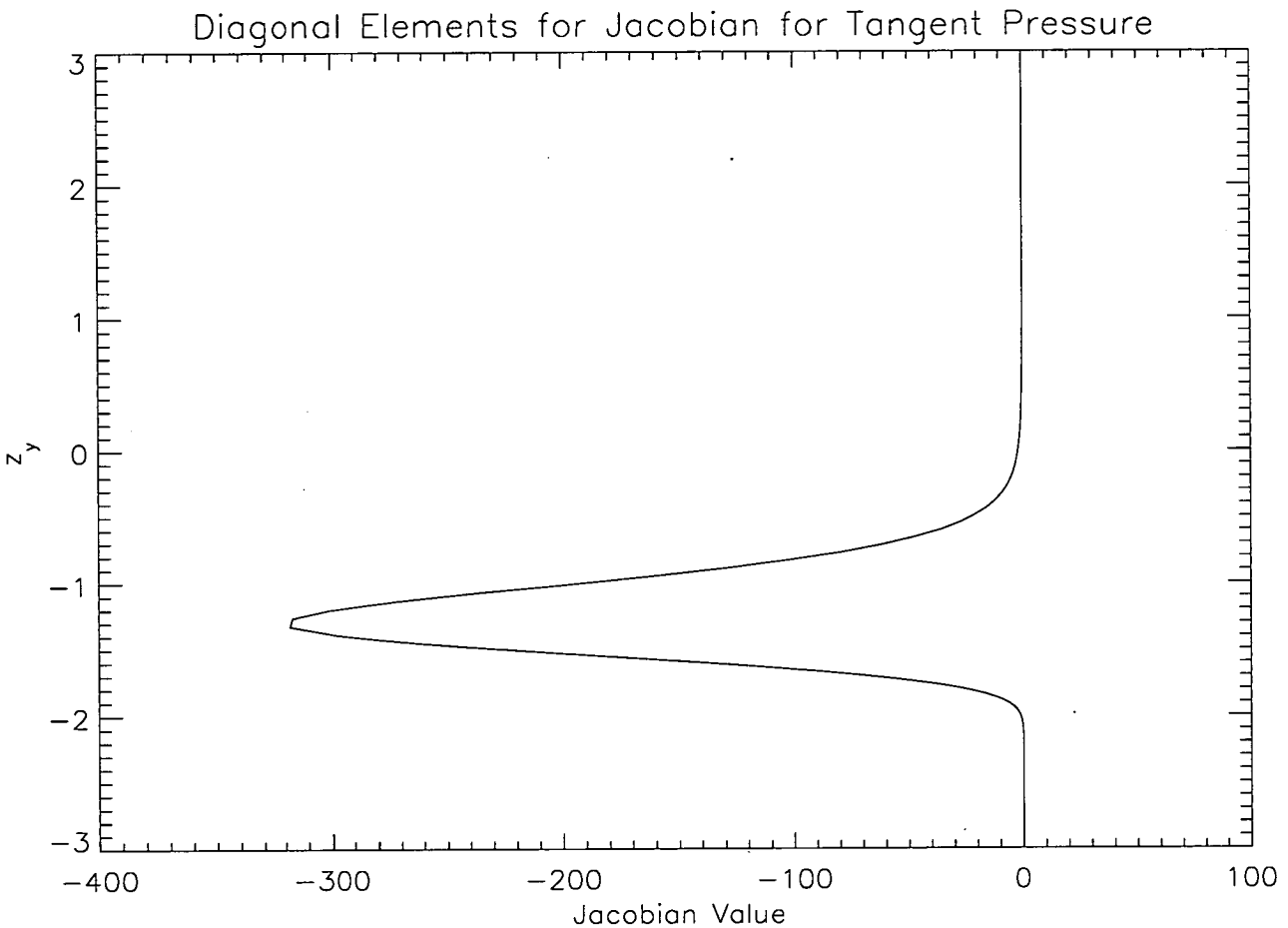

Figure 6.11: The non-zero elements of the K matrix for tangent pressure used by the traditional forward model.

K matrix. This implies that radiances at one level do depend on the tangent pressures at other levels. Summing all the tangent pressure contributions for each radiance output results in figure 6.13. For reference, the truth (red line) and the difference (dashed line) are plotted as well. It can be seen from this that the overall influence function is correct (the largest error is around 5%). This means that the neural network expects each tangent pressure to influence a group of radiances, not a single radiance, however, the network is using the total tangent pressure information across each radiance as expected. Figure 6.12 shows the same overall structure as the temperature section of the Jacobian. This suggests the network is unable to separate the contributions from the tangent pressures on the radiances from the effects of temperature.

In the reduced neural network trained in section 6.4, there is only one tangent pressure input, resulting in a single element for the tangent pressure K matrix. This has the value of -315.11, compared to the true value at that level of -316.80. This shows that when other tangent pressures are not present, the neural network can calculate the derivative of the radiance with respect to the tangent pressure well. It may thus be possible to improve the use of tangent pressures within the neural network using a more complex network structure, however this was not investigated as the radiances generated are within instrumental noise and the tangent pressures are not part of the assimilation scheme.

Previously, it was stated that the tangent pressures cannot be part of the assimilation model's state vector due to technical reasons. In this section, it was shown that, in its current configuration, the neural network cannot produce an accurate enough Jacobian for tangent pressures. Without an accurate Jacobian, tangent pressures cannot be used within the assimilation process. This supports the previous argument for retrieving the tangent pressures separately from the forward model and assuming them to be known.

6.6 Jacobian for Constituant Species

So far, this chapter has examined the Jacobian for temperature and tangent pressure. In chapter 5, the neural network was extended to include additional species in the forward model calculation. In this section, the Jacobians for these species is examined. Chapter 5 used band 7, a band centered on an ozone line,

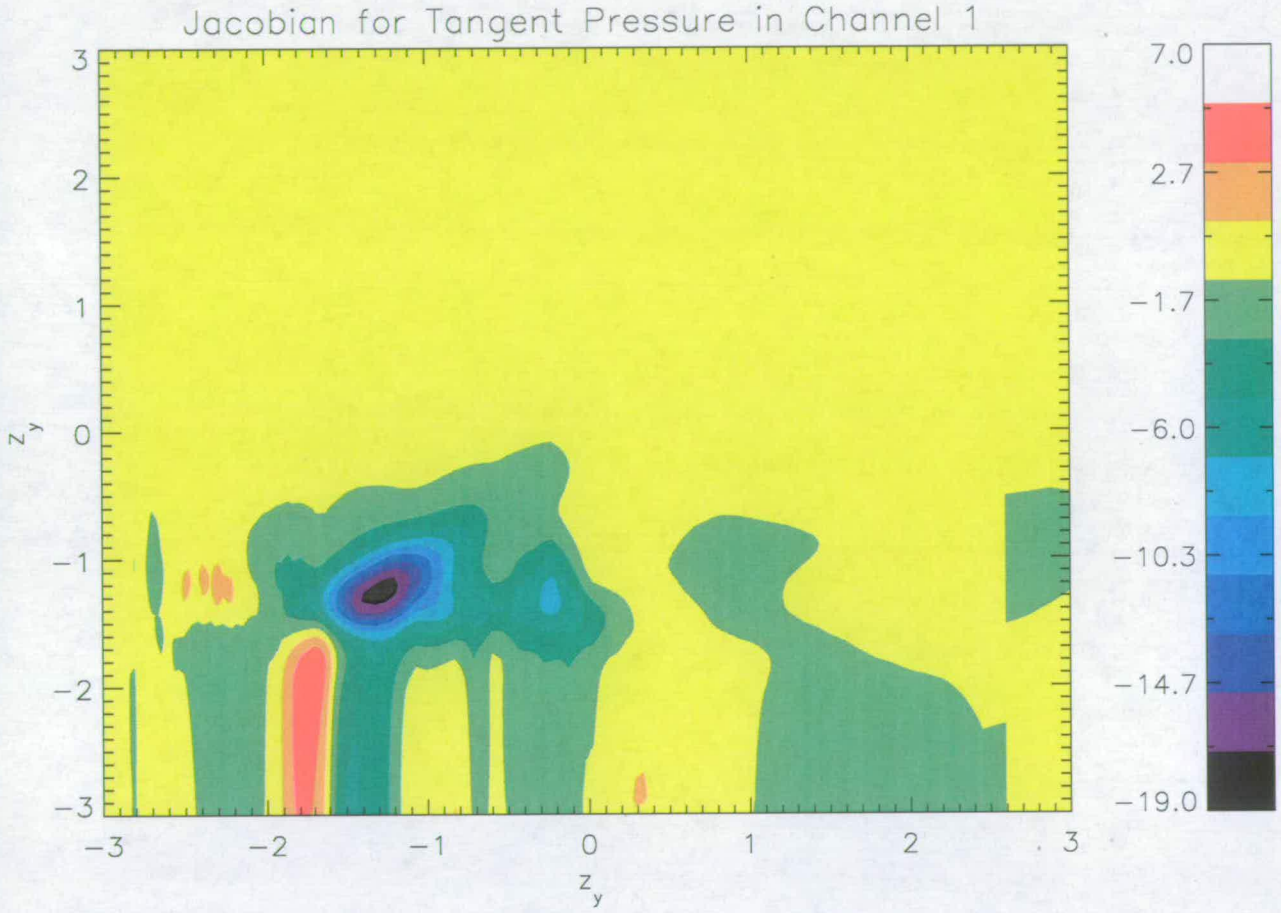

Figure 6.12: The Jacobian for the network with respect to the tangent pressures for channel 1. Ideally, this should be sparse matrix, with the z_y only being affected by the corresponding tangent pressure. Here, this is not the case, implying that each radiance depends on more than one tangent pressure.

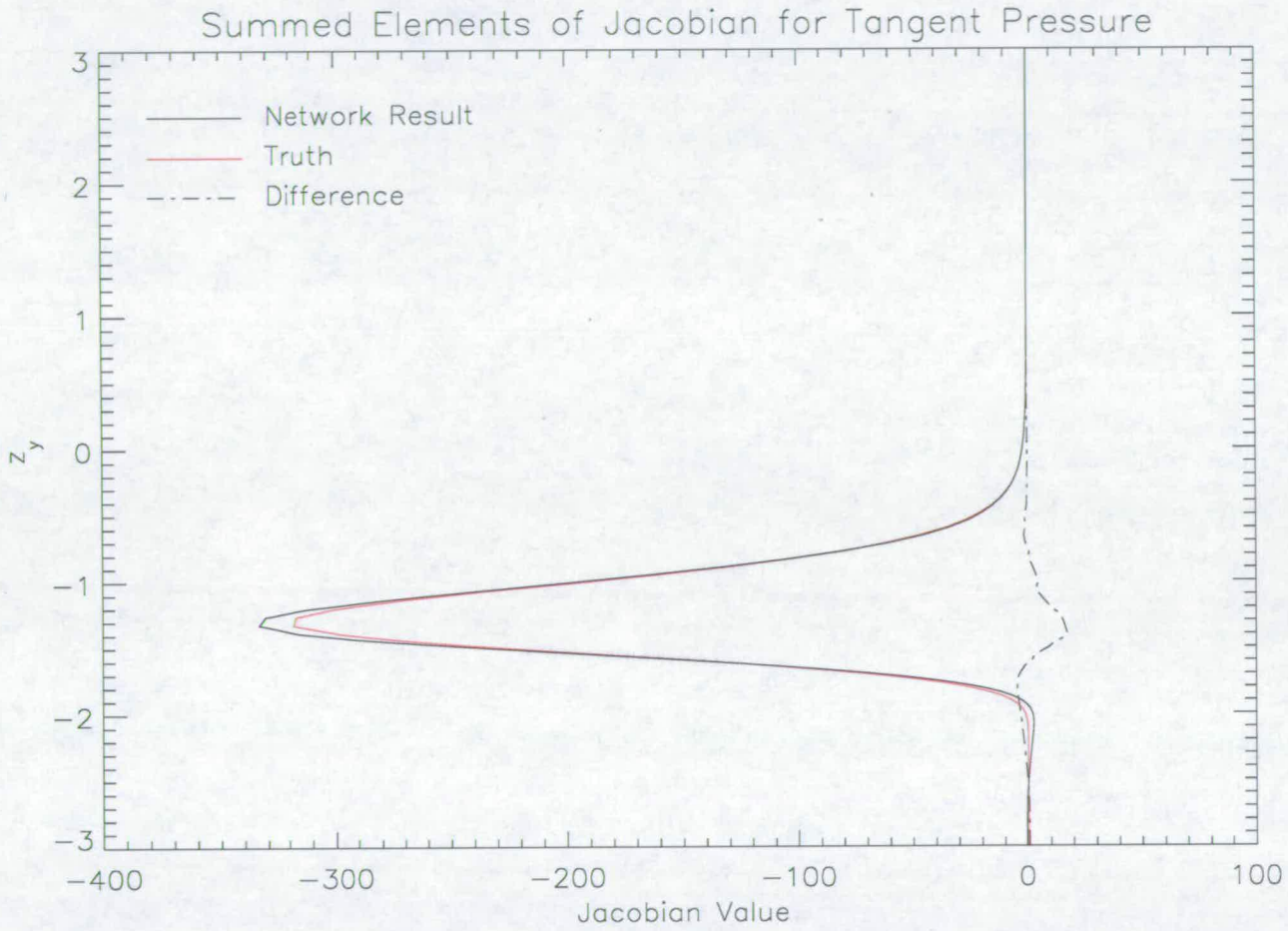

Figure 6.13: Summing the tangent pressures along each output level leads to the correct influence function. This shows that the network is using the tangent pressure information in the expected way but expects each radiance to affect those around it.

for this work. Here, the same band will be considered.

The neural network for band 7 relied on ozone and water vapour as inputs and ignored temperature information. It was found that above about $150hPa$, the neural network performed well, but below this the network produced large errors in the results. Here, the Jacobian of these inputs is examined and possible reasons for this poor performance are considered.

The derivatives are calculated in the same way as previously explained. Here, the neural network is using sigmoid activation functions. In band 1, this produced unreasonably large errors in the Jacobian. The Jacobian for the true forward model with respect to ozone is presented in figure 6.14. Here, the radiances are solely affected by the ozone concentrations near the measurement height. Figures 6.15 and 6.16 show the Jacobian generated from the neural network and the difference from truth.

Below $z_y = 0.3$ (around $0.5hPa$), the Jacobian from the neural network is very similar to the true Jacobian, with only small deviations near the bottom of the atmosphere. Above $z_y = 0.3$, the neural network Jacobian drops by an order of magnitude compared to the true Jacobian. Figure 5.18 in chapter 5 shows that the radiances above $z = 0.3$ are effectively at background radiation level and hence contain no useful information.

Previously, it was found that the neural network performs poorly in the lower regions of the profile. The reason for this poor performance was thought to be due to the effects of water vapour (see chapter 5). If the neural network was unable to cope well with water vapour inputs, this is the region the problem would show in. The Jacobian for water vapour, generated using the true forward model is shown in figure 6.17. Here, the scale has been changed to concentrate on the lower part of the atmosphere. Outside this region, the Jacobian is very close to zero.

Figure 6.18 shows the corresponding region of the neural network generated Jacobian. As can be seen, the main peak (at $z_y = -2.5$) has got a similar shape to the true Jacobian but its effect is an order of magnitude lower. Beyond this ($z_{H2O} > -2.4$), the values rapidly increase in magnitude. Figure 5.19 in chapter 5 shows a typical water vapour profile. It can be seen that the largest concentration of water vapour is below $z < -2.4$, where the network Jacobian follows the true Jacobian in shape. Above $z = -2.4$, the concentration of water vapour drops

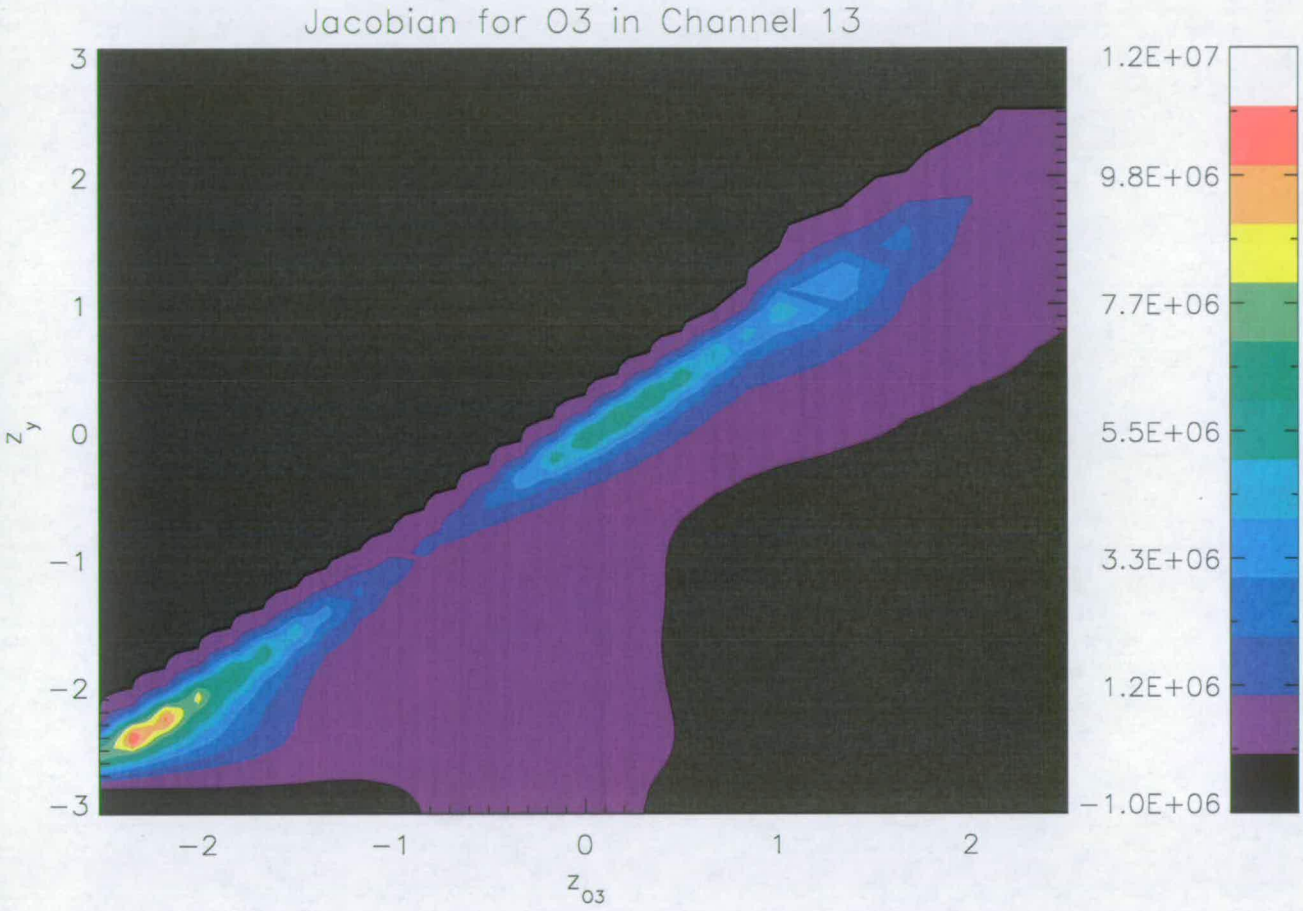

Figure 6.14: The real Jacobian for ozone in channel 13 of band 7, generated using automatic differentiation of the real forward model. This shows that the radiances are almost entirely affected by the ozone concentrations at the measurement height.

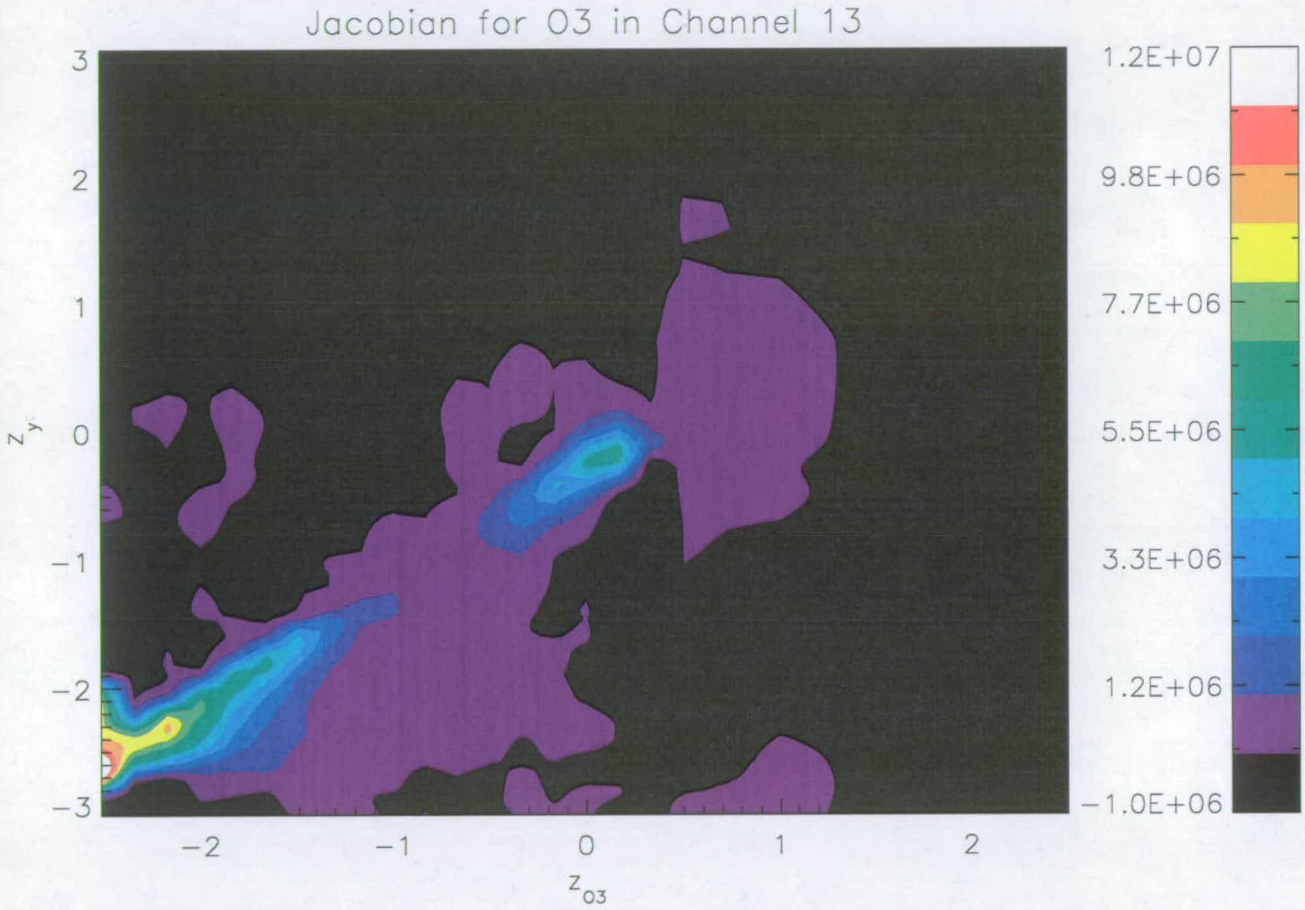

Figure 6.15: The neural network-generated Jacobian for ozone in channel 13 of band 7. Compared to figure 6.14, the network performs well up to around $z_y = 0.3$ $(0.5hPa)$ where the Jacobian becomes much smaller-valued.

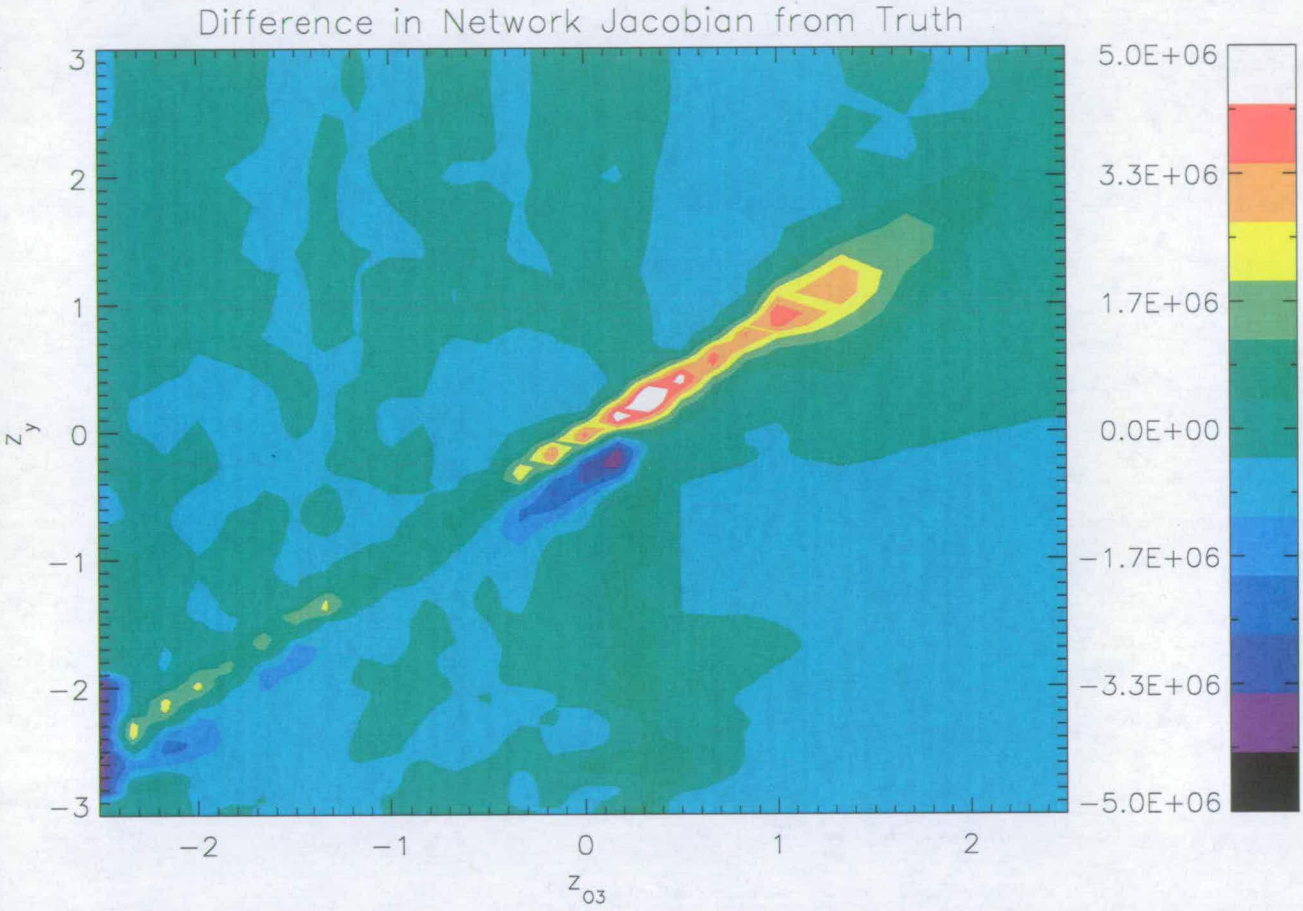

Figure 6.16: The difference between the neural network-generated Jacobian (figure 6.15) and the true Jacobian (figure 6.14). As expected above $0.3hPa$, there are large differences where the neural network uses less ozone information.

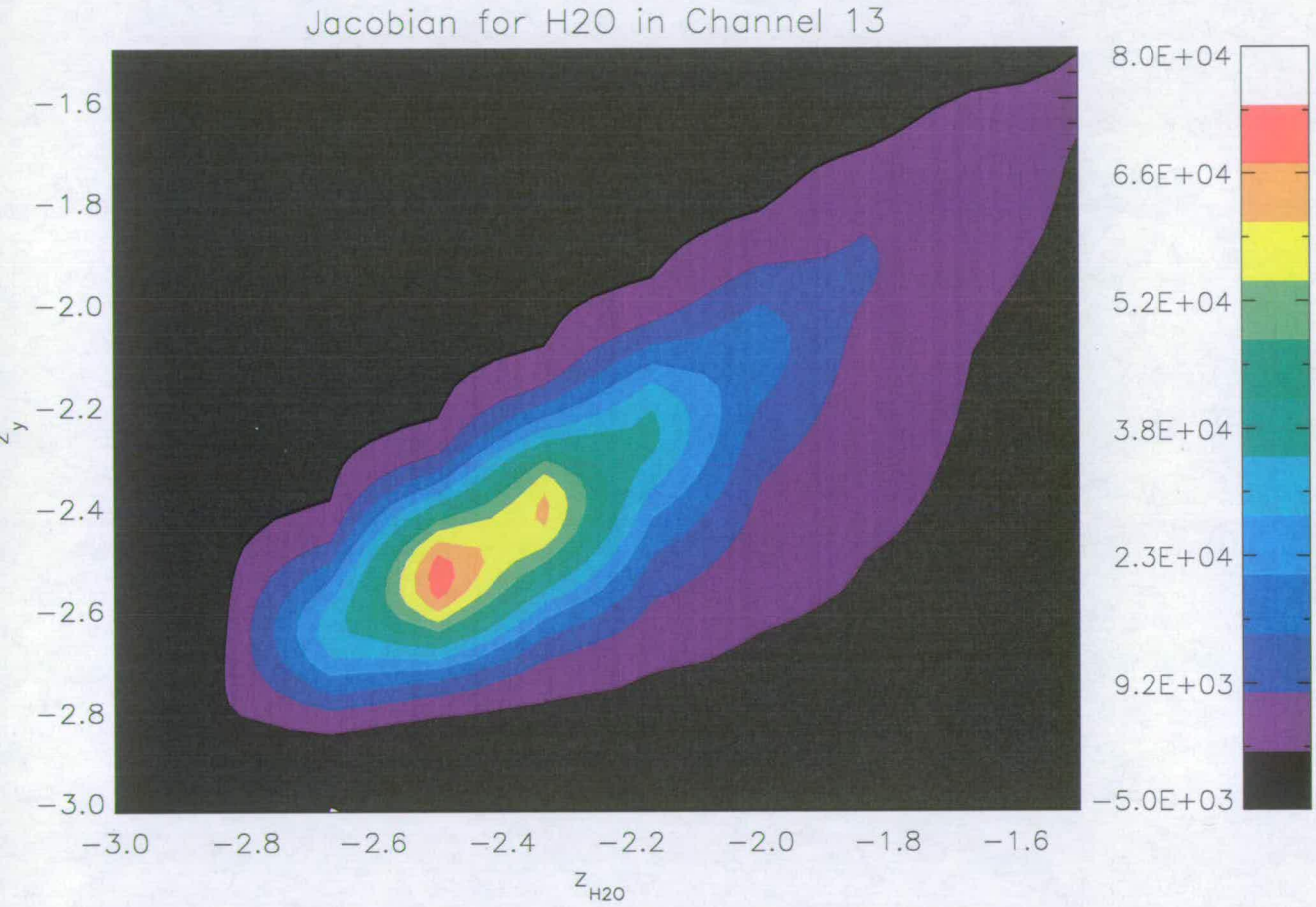

Figure 6.17: The true Jacobian for water vapour in channel 13 of band 7. The scale has been changed to only show the lower part of the atmosphere, where the water vapour has an influence.

significantly. The true forward model ignores water vapour information above $z = -2.4$ (i.e. its Jacobian is zero at that point) but in the neural network, these values contribute to the radiances. Even if the contribution is small, the effect seen in the Jacobian will be large, due to the concentration of water vapour being small. This is what is seen in figure 6.18.

6.6.1 Improving the Neural Network Performance

One possible solution to this problem might be to split the problem across two networks. The first would cover the lower part of the radiance profile (where water vapour affects it) and use water vapour and ozone profiles as inputs. The second network would cover the upper part of the profile and only use ozone profiles as inputs. To test this, two networks were trained, the first ran the upper 95 minor frames of the profile (network A) and the second trained on the lower 30 minor frames (network B). As expected, network A produced similar errors to the previous results. Network B did improve from $\sigma = 1.4K$ to $\sigma = 1.0K$, still well above the instrument noise level.

There are several improvements that could be made that might reduce the error but these were not tested due to lack of time. One possibility is to introduce skip layer connections (e.g. Ripley (1997)) between the input ozone levels and the radiances at the corresponding tangent heights. As shown previously, only the ozone concentrations very near the measurement height have an affect on the radiances. Doing this would reduce the complexity of the problem for the neural network as it does not have to deal with irrelevant information in each output node.

6.7 Discussion

This chapter has shown that it is possible to calculate the Jacobian for the neural network-based forward model using analytical differentiation. The Jacobian for temperature was investigated for two channels. In the first, it was found that the Jacobian may be acceptably accurate for use in an assimilation scheme, though testing within the assimilation environment would be necessary. In the second channel, the Jacobian was less accurate, though may still be acceptable in an assimilation scheme. A reduced neural network was constructed to examine the

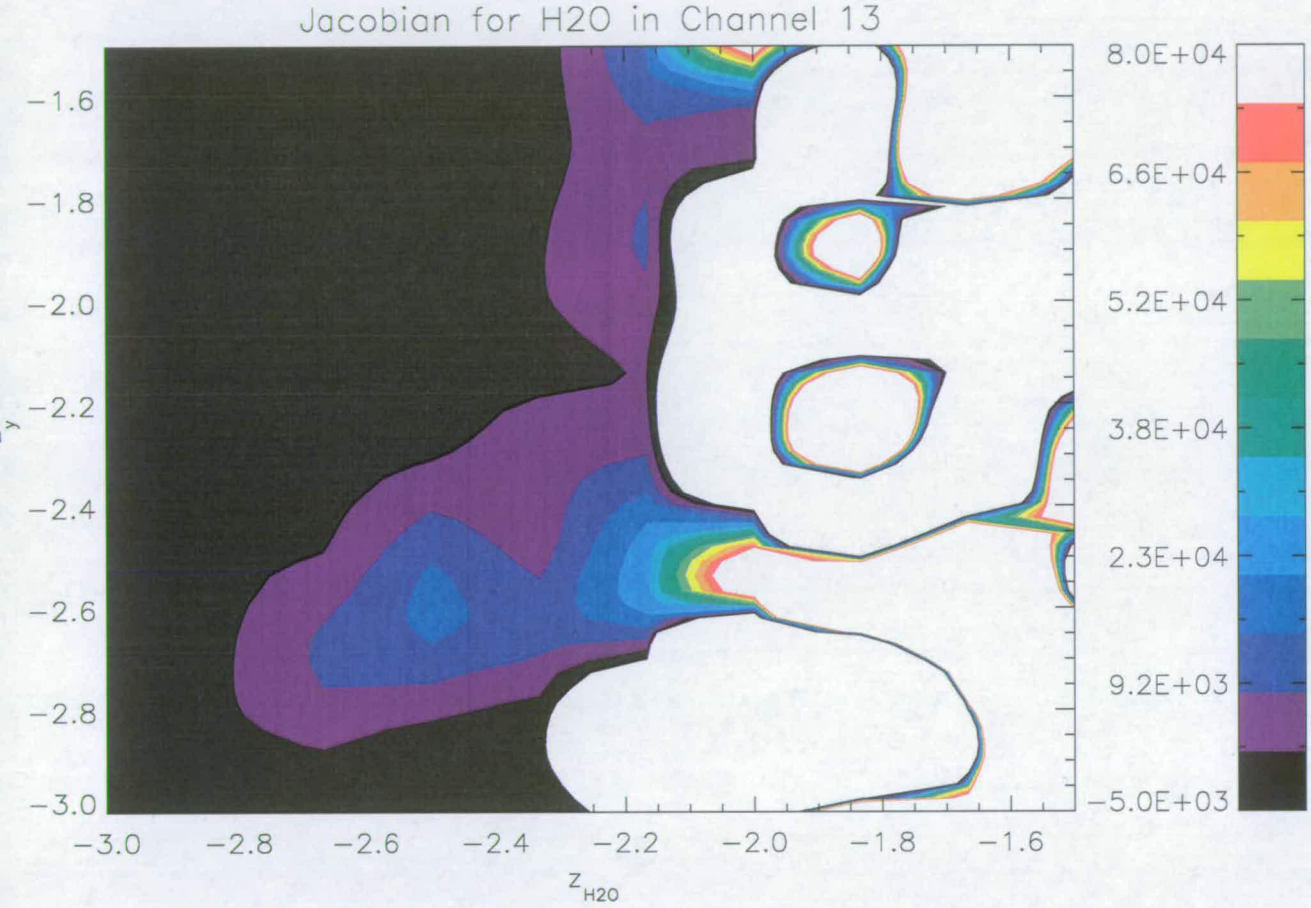

Figure 6.18: The neural network-generated Jacobian for water vapour in channel 13 of band 7. Compared to figure 6.17, the main feature is the right shape but there are large areas where the Jacobian does not fit in the scale. This is partly due to the scarcity of water vapour in the atmosphere above $z_{H2O} = -2.5$. Small changes above this height will result in dramatic changes in the measured radiance.

Jacobian for a single minor frame, where the error in the Jacobian was greatest. In this reduced network, the Jacobian was very close to the true Jacobian. This suggests it is possible to improve the Jacobian across the entire profile.

It was found that using sigmoid transfer functions in the neural network, in this case, produced significant errors in the Jacobian. Instead, the networks for the two channels examined were retrained using hyperbolic tangent transfer functions. It is thought that hyperbolic tangent transfer functions allow the network to more easily find the global minimum in weight space, hence allowing more accurate Jacobians to be found.

The Jacobian with respect to the tangent pressures was also investigated and found to be much different from the true forward model. This implies that the "black box" nature of the neural network is using the tangent pressures in a way that the true forward model does not use them. In the reduced neural network, the tangent pressure Jacobian was found to be very close to the true value (less than 1% difference). This again suggests that the Jacobian can be improved.

Possible ways of improving the neural network include looking at using skip-layer connections and investigating more advanced training methods. As the tangent pressure Jacobian is sparse, with each tangent pressure only affecting its corresponding minor frame radiance, using skip-layer connections could be constructed between the tangent pressure inputs and their own corresponding radiance output. This may allow a more accurate Jacobian for tangent pressures to be created. It may also help reduce the errors seen in the temperature Jacobian.

Chapter 5 examined the use of a neural network in a band dominated by an ozone line. Here, the corresponding Jacobian was examined. It was shown that the Jacobian generated from the neural network was very similar to the true Jacobian up to 0.3 hPa. Above this, the Jacobian is an order of magnitude smaller than the true Jacobian. Above this height, the radiances in the channel had values very near background radiation levels and would not be used in an assimilation scheme.

The Jacobian for water vapour in this channel was also investigated. It was found to differ significantly from the true Jacobian. This is the reason the neural network had difficulty modelling radiances in the lower atmosphere in this band. Several suggestions have been made to improve this but due to the complexity of the changes required in the code and a lack of time, these were not tested.

Calculating the Jacobian using analytical differentiation comes at little cost within a forward model run. The intermediate variables are already available from the initial run of the neural network and all that is required is a series of additions and multiplications. The analytical differentiation produces identical results to perturbing the network inputs, but is substantially faster.

Overall, the network Jacobians generated in this chapter for temperature and ozone may be acceptable in an assimilation model, though testing within the assimilation environment is necessary to determine this. The Jacobian for water vapour was shown to be largely inaccurate and in need of further work. Several suggestions for further work have been made, should the Jacobians prove unacceptable.

Chapter 7

Conclusions and Discussions

The work carried out in this thesis has demonstrated that it is possible to construct a forward model for the EOS-MLS based on neural networks. It has been shown that a neural network can perform well in band 1, which is centered on an oxygen line so temperature and pressure have the largest effect on radiances. It has also been shown that the Jacobian for this band, calculated by analytical differentiation of the neural network, may be acceptable in an assimilation scheme but testing within the assimilation environment would be needed. It was further shown that discrepancies in the network Jacobian can be overcome in principle.

The issue of how to cope with tangent pressures in an assimilation scheme has also been examined. As has been stated, the assimilation model does not have tangent pressures in its state vector and these are unlikely to be added for technical reasons (Feng (2004)). It has been shown that these tangent pressures can be retrieved, outside the assimilation scheme, using a neural network with errors that are comparable to traditional retrieval methods.

The main reason for investigating the use of a neural network as a forward model is computer time. Assimilating instrument measurements takes a large amount of computer power and anything to reduce this would allow the computer time to be spent on other tasks, such as increasing the number of instruments assimilated or increasing the resolution of the model. As computers become more powerful, instruments also become more complex and require more computing power to run their forward models. This means there will always be a need to reduce the processing cost of the forward models.

Neural networks provide such a way. Table 7 shows the times needed to

run 10 profiles through the neural network and the true forward model for one channel. Both runs were carried out on a SunBlade 100 desktop PC running at 502MHz with 256MB of RAM. Each run was carried out twice to reduce the effects of network latency. Although this well below the power available to run the assimilation process, it can be seen that the neural network is almost 100 times quicker than the full forward model while still having acceptable errors. As linearised forward models are currently unavailable, no testing compared to these could be carried out.

Model	*Run 1*	*Run 2*
True without Jacobian	2m 31.1s	2m 30.3s
True with Jacobian	13m 52.7s	13m 47.0s
Neural Net without Jacobian	2.1s	1.9s
Neural Net with Jacobian	9.4s	8.7s

Table 7.1: A comparison of running times between the neural network and the true forward model

Chapter 2 gives a list of prerequisites for incorporating measurements into a 4D-VAR scheme. The first is a fast forward model. This has been shown to be achievable using neural networks (approximately 100x faster than traditional forward models). The second thing required is the Jacobian for the forward model. Chapter 6 has shown that the Jacobian for a neural network can be calculated using analytical differentiation. For the large majority of the profile, the errors in the Jacobian are small but near the largest values, the discrepancies in the Jacobian become larger. As shown in chapter 6, these Jacobians may be acceptable, subject to testing within the assimilation environment. If these discrepancies are a problem, more work needs to be done to improve the accuracy of the Jacobian. The final thing needed is an estimate of the error covariance matrix for the instrument. This should include instrument errors, interpolation errors and errors due to the forward model. Here, the forward model contributes errors typically around $\sigma = 0.1K$ (around 1/3 of the instrument noise). The testing phase of the neural network provides a number of radiance profiles that can be compared to the equivalent profiles in the testing set. This, combined with characteristics from the assimilation model, should provide a good estimate of the error covariance matrix for the neural network forward model.

In order to use a neural network forward model in an assimilation scheme, the following steps must be taken prior to including it:

1. Decide which minor frames from which channels / bands will be used in the assimilation scheme
2. Generate training set from real forward model based on these, covering all expected input values
3. Train the neural networks to generate weights for forward model
4. Train a neural network to retrieve tangent pressures

Once these have been done, it is possible to use the neural network in place of the full forward model.

There are several limitations on neural networks. The major disadvantage is that they handle poorly inputs which are outside their operating range. This means that all inputs must be checked to ensure they lie within the expected ranges and if not, either discard the profile or run it with a full forward model. Another disadvantage is that a neural network is unable to run new channels without first being trained for them.

The work in this thesis has dealt with training data generated using a non-tomographic forward model, i.e. the atmosphere is considered horizontally homogeneous. In reality, the radiances are affected by inputs across a large (horizontal) area, over which the atmosphere is likely to change significantly. As was stated in chapter 3, this non-tomographic forward model reproduces the true radiances within approximately $1K$. To improve this, data from several profiles are used within a tomographic forward model. It would be possible to simulate this in a neural network by increasing the number of inputs in the neural network to accommodate more input profiles. In this case, the size of the training and validation sets may need to be increased to cover a much larger range of conditions.

There are several ways the work in this thesis could be extended. The first and most obvious way would be to extend the network to work in other bands. Assimilation models are starting to deal with more chemical species than just ozone and there are chemical transport models that are already implementing data assimilation. The EOS-MLS and other satellite data contain a lot of information about these species and could be useful in their assimilation processes.

The work in this thesis has mainly concentrated on band 1 of the EOS-MLS. This is because the effects of the oxygen line at 118.75 GHz dominate the radiances in this band, while the upper sideband is masked. It has been shown that in all channels of this band, the neural network-based forward model works well. Band 7, a highly non-linear band centered on an ozone line was also considered. In this case, the radiances in the lower atmosphere (below $z = -2.2$) have large errors when generated using the neural network-based forward model. Above this, the radiances are well below instrument noise levels. This suggests that the work here should apply to other bands, provided the appropriate species are included in the input state-vector.

Another possible improvement might be to investigate other training methods. Here, backpropagation was used, while quickprop was found to be unsuitable for this network. There are a large number of other training methods that could be investigated such as Bayesian learning (e.g. MacKay (1995)). These more advanced training methods may significantly improve results and help resolve some of the outstanding issues discovered during the course of this work (such as improving the Jacobian).

There are several other possible fast forward models for the EOS-MLS currently in development, principally a linearised and a quadracised forward model being developed by Feng (2004). These operate by assuming the radiances have a near-linear (or near-quadratic) dependency on the model inputs around the mean value which allows the forward model calculation to be greatly simplified. Currently, these forward models operate non-tomographically[1] and achieve radiances well within instrument noise levels for several bands.

As these models are based on traditional forward model techniques, they are easy to extend to other bands of the instrument. One problem with a linear forward model arises when the dependency between inputs and radiances is not linear enough, such as in band 7 of the EOS-MLS. In this case, the resulting radiances will have large errors. Errors can also occur in near-linear bands when the inputs are far away from the mean value, when the deviation from linearity becomes larger.

Neural networks are inherently non-linear and hence can avoid large errors when the radiances are not linearly related to inputs. This can be seen in chapter

[1]Current work on these forward models include expanding them to work tomographically

5, where the neural network-based forward model was extended to band 7 - a highly non-linear band. Although there were large errors below $z = -2.2$, above this the neural network was well trained. When the inputs are far away from the mean value, the neural network may suffer from increased errors due to the normalisations that are applied.

Overall, it is felt that neural networks provide a viable alternative to traditional forward models in this case but some work must be done before they are able to be used in a real assimilation scheme. In addition, it has been shown that tangent pressures can be successfully, and rapidly, retrieved using a neural network, independent of the forward model used within the assimilation scheme.

Appendix A

Further Discussion of Neural Networks

This appendix continues the discussion of neural networks from chapter 2 to provide illustrations of the major type of neural networks used within this thesis. It is intended to give readers unfamiliar with neural networks a better grounding for following discussions in this thesis. The discussion begins by considering the simple perceptron case (where there are no hidden layers) and illustrates how this type of network is run and trained. It then considers the hidden-layer perceptron case, the major type of neural network considered in this thesis and illustrates how these are run and trained.

A.1 Simple Perceptron

This section looks at no-hidden-layer perceptrons and describes how they work. No-hidden layer perceptrons are the most basic non-trivial neural networks. Their inputs are directly connected to their outputs, as illustrated in figure A.1. When run, the input values are multiplied by the weights connecting the input node to the output node and summed. The resulting value is then "activated" using the activation function (equation A.1 in this case), producing the output value.

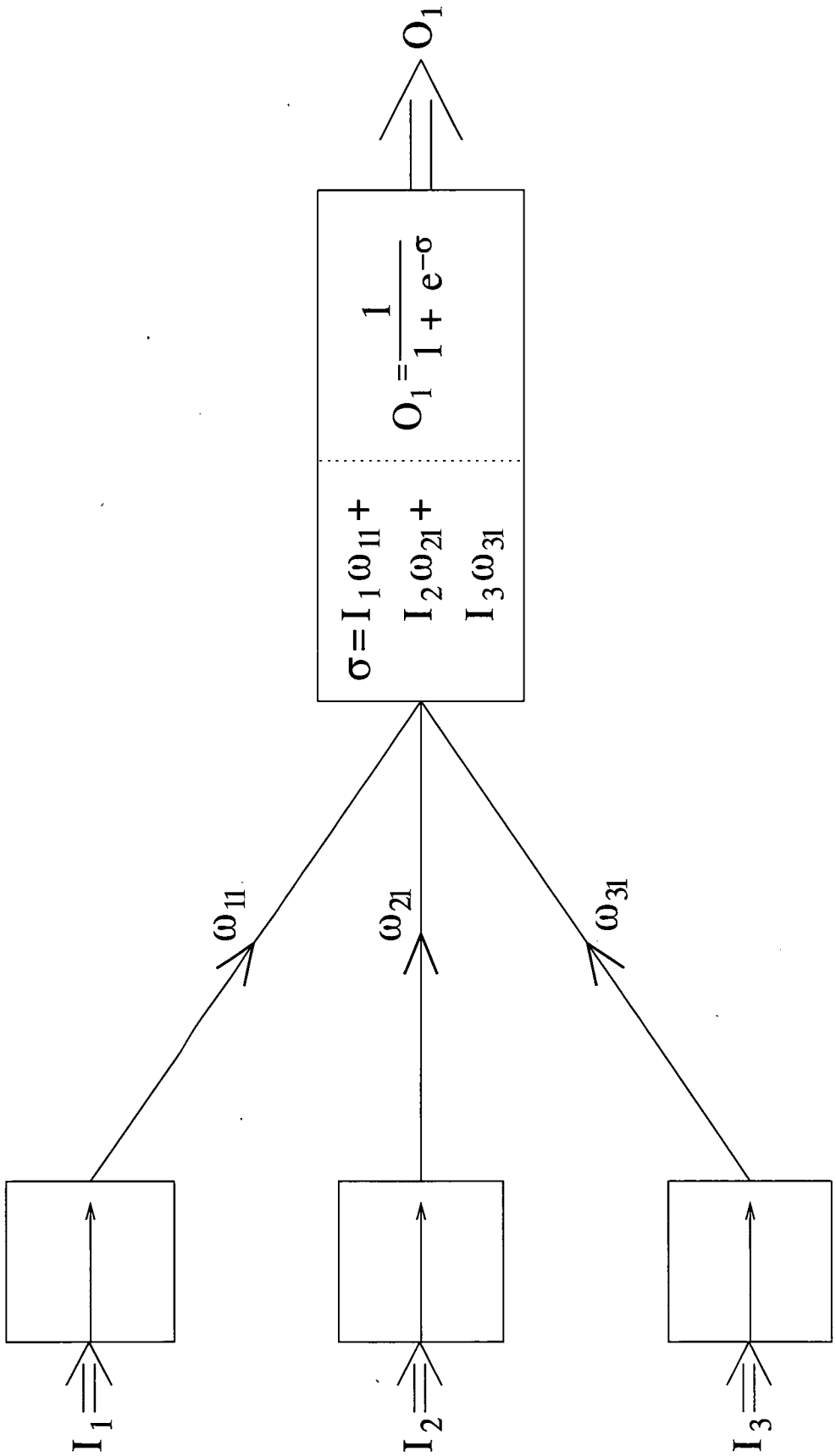

Figure A.1: The layout of a simple (no-hidden-layer) perceptron. Inputs are passed through the input layer, where normalisations are applied. They are multiplied by the appropriate weights (connecting lines) and summed to produce σ. This σ value is then "activated" to produce the final output value.

$$V(\sigma) = \frac{1}{1 + \exp -\sigma} \qquad \sigma = \sum_i \omega_i I_1 \tag{A.1}$$

The "weights" in the network can be found by a process of training. To train a network, a representative group of input-output vectors are found by other means. From this set of profiles, one profile is chosen at random and run through the network and the error calculated against the true output value. This can then be used to update each weight in the system using equation A.2, where E_i is the error on the output compared to the true value (d_i), taken from the training set. The process of selecting a random profile from the training set and updating the weights is then repeated until the network is considered fully trained.

$$E_i = (d_i - O_i)^2$$
$$\omega_i(new) = \omega_i(old) + \eta E_i I_i \frac{dV(\sigma)}{d\sigma} \tag{A.2}$$

A network can be considered fully trained using a number of different criteria. The most common is to have a "validation set". This is a selection of input-output profiles, separate from the training set that are run through the network periodically (though the network is not trained on these). The error across the entire validation set (the "validation error") is then recorded. If the validation error is lower than previous validation runs, he internal weights of the system are stored. Once the network has completed a pre-defined number of validation runs without improving its validation error, the network is considered fully trained and the system weights are restored to those that produced the best validation run.

A.2 Multi-layered Perceptrons

As discussed in chapter 2, single-layer perceptrons are limited to solving "linearly separable" problems. To remove this restriction additional, hidden, layers of nodes are introduced in the network, as shown in figure A.2, which shows the steps

involved in running a multi-layered perceptron. Training a multi-layer perceptron is more difficult than training a simple perceptron as there are now multiple layers of weights that must be updated during training, with no direct measure of error available for the hidden node values.

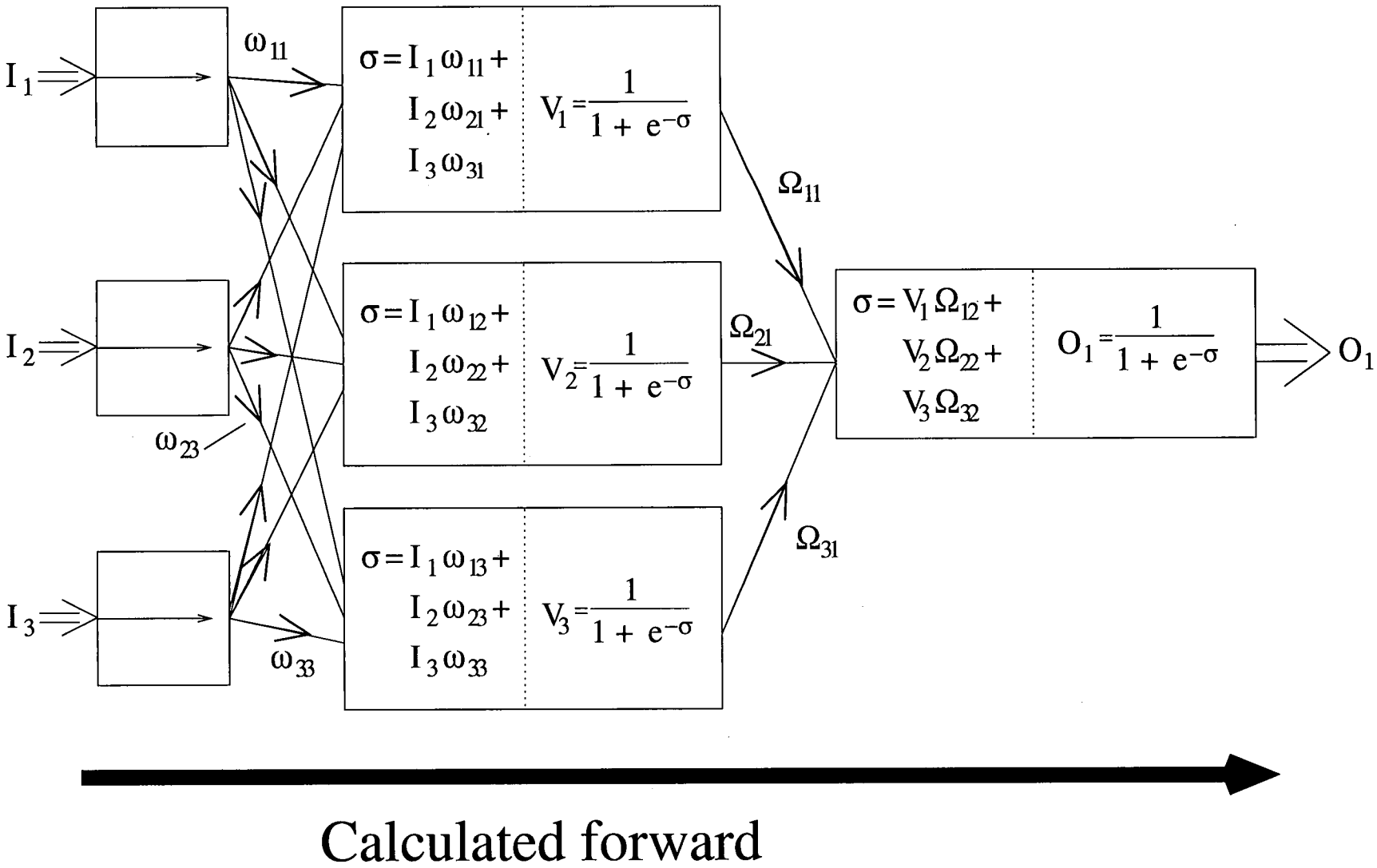

Figure A.2: The layout of a perceptron with one hidden layer. Inputs are passed through the input layer, where normalisations are applied. They are multiplied by the appropriate weights (connecting lines) and summed to produce σ within the hidden nodes. This σ value is then "activated" to produce the output for the hidden nodes (V). The output from the hidden nodes are then passed along to the output layer where the same process occurs again.

Backprop provides a way of estimating the portion of the final value error coming from the input-to-hidden weights as well as the hidden-to-output weights. As discussed in chapter 2, a "sensitivity" factor, δ is calculated from the final error for each output node, defined by equation A.3. Sensitivity factors for each hidden node can then be calculated using equation A.4. The weights can then be updated using equation A.5- A.6 (for the hidden-to-output weights and input-to-hidden weights respectively). Figure A.3 shows the steps involved in updating the weights (the so-called "backpass" of backprop).

$$E_i = (d_i - O_i)^2$$
$$\delta_k = \frac{dO(\sigma)}{d\sigma} E_i \tag{A.3}$$

$$\delta_k = \frac{dV(\sigma)}{d\sigma} \sum_h \Omega_{kh} \delta_h \tag{A.4}$$

$$\Omega_{kh}(new) = \Omega_{kh}(old) + \eta \delta_h O_k(\sigma) \tag{A.5}$$
$$\omega_{kh}(new) = \omega_{kh}(old) + \eta \delta_h V_k(\sigma) \tag{A.6}$$

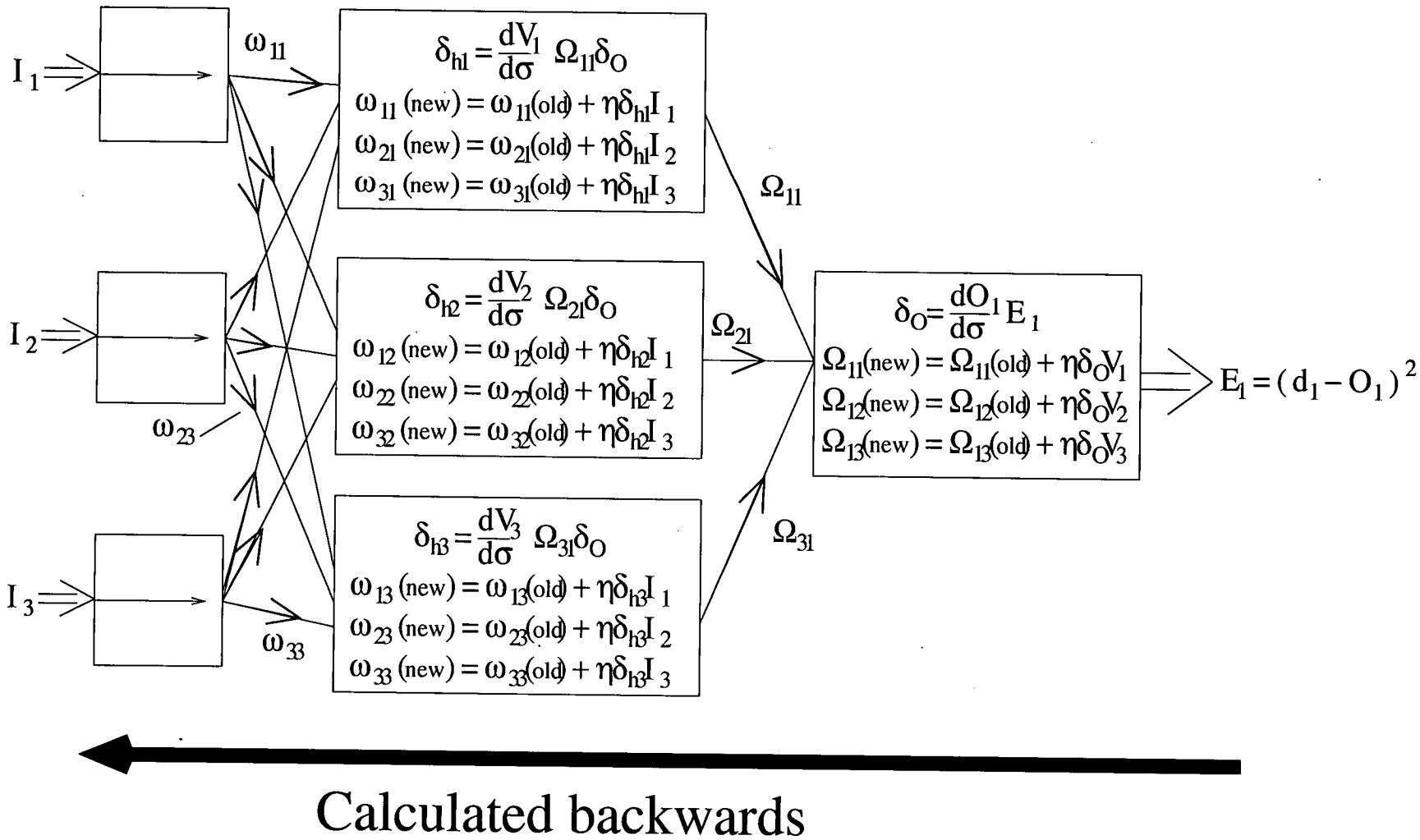

Figure A.3: The steps involved in training a hidden-layer perceptron. Initially, the error in the output is calculated and a "delta" value (δ_O is computed for each output node. This δ value is then combined with the "upstream" weight and the derivative of the hidden node to produce the hidden nodes delta value (δ_{hi}) Using these delta values, the new weights can be calulated as shown.

Appendix B

Definitions

Quantity	Definition
EOS-MLS	
EOS	Earth Observation System
MLS	Microwave Limb Sounder
EOS Aura	Satellite the EOS-MLS instrument is on
MIF	Minor Frame
MAF	Major Frame
DACS	Digital Autocorrelator Spectrometer
P / T	Pressure / Temperature
UARS MLS	The predecessor to the EOS-MLS
UARS	The satellite the UARS MLS was flown on
FOV	Field Of View
Neural Networks	
NN	Neural Network
Node / Neuron	The basic calculation unit
Layer	A collection of nodes. Typically a network is made up of an input layer, hidden layer(s) and an output layer
Committee	A group of networks
Weight	Strength of a link between nodes
Training phase	The phase of network evolution where the weights are changed
Validation Phase	The phase of network evolution where the network is tested to see if it has improved over previous validation phases

Continued on next page

Table B.1 – continued from previous page

Quantity	Definition
Testing Phase	The phase of network evolution after training is complete where the network's ability to generalise is tested
Upstream	Refers to the previous layers - the layers closer to the input layer
Downstream	Refers to the layers closer to the output layer
ADALINE	The simplest type of neural network. One binary neuron that performs a threshold transfer of inputs
MADALINE	A committee of ADALINEs that output binary values depending on a majority vote
Perceptron	A general class of neural networks that form the basis of most computational neural networks
Backprop	A method of training neural networks that relies on the derivatives of each node to update their weights.
Quickprop	A method of training neural networks that relies on both the first and second derivatives of each node to update their weights
Epoch	The number of validation runs performed during training
Data Assimilation	
ECMWF	The European Centre for Medium Range Weather Forecasting
4D-VAR	4-D variational assimilation - A type of assimilation process
Other	
SSM/I	Special Sensor Microwave / Imager
OMBFM1	A neural network for the SSM/I based on a neural network
NWP	Numerical Weather Prediction
GPH	Geopotential Height
GCM	General Circulation Model
Symbols	
$I(\nu)$	Intensity per unit area
ν	frequency
τ	Optical Depth
$k(\xi, \nu)$	Total absorption Coefficient, defined in terms of volume
T	Absolute temperature
k_B	Boltzmann Constant

Continued on next page

Table B.1 – continued from previous page

Quantity	Definition
c	Speed of light
h	Planck constant
λ	Wavelength
ξ	Distance along observation path
P	Pressure
ρ	Density
g	Acceleration due to gravity
h	Geometric Height
H	Geopotential Height
M	Mole mass of a gas
R_u	Universal gas constant
z, ζ	Pressure coordinate, $z = -\log_{10}(p)$ (used interchangeably)
W_i	Weight in an ADALINE / MADALINE network connecting input i to the output node
E_i	Error n an ADALINE / MADALINE network.
d	Desired output for an ADALINE / MADALINE network
η	The learning rate for a neural network
I_i, $I_n(i)$	Input i into a neural network after normalisation
$I_u(i)$	Input i into neural network before normalisation
O_i, $O_n(i)$	Output i from a neural network after normalisation
y_i, $O_u(i)$	Unnormalised output i from a neural network
$\Delta\omega(t)$	Weight change at time-step t
$S(t)$	$\frac{\delta E}{\delta \omega}$at time-step t
a_i	Normalisation multiplicative factor for a neural network
b_i	Normalisation additive factor for a neural network
d_i	Desired output i of a neural network
ω_{ij}, Ω_{ij}	A weight from node i to node j
σ	The summed inputs into a node
$ac(\sigma)$, $\phi(\sigma)$, $\gamma(\sigma)$	The activation function with respect to σ ($\phi(\sigma)$ and $\gamma(\sigma)$ are used to indicate different activation functions within the same network)

Continued on next page

Table B.1 – continued from previous page

Quantity	Definition
δ_i	The sensitivity factor for node i
V_k, z_i	The outputted value of node k ($= O_i - d_i$ for output node, i)
α	Momentum coefficient for a neural network
ν	Weight decay coefficient for a neural network
μ	Maximum growth factor in quickprop
B_i, β_i	Bias on node i
$\vec{x}_k$	The model state vector at time step k
$\vec{x}_k^b$	The background model state vector at time step k
$\vec{x}_k^a$	The analysis model state vector at time step k
$\vec{u}_k$	Model inputs vector at time step k
$\vec{y}_k$	Observation vector at time step k
$\vec{\lambda}_j$	The adjoint equation, j
B_0	Covariance matrix of initial background model state error
K	Gain matrix
H	Observation matrix, including forward model and grid interpolations
F, G	Model forcing matrices
$h_k(\vec{x_k})$	Observation function. Analogous to H
$\vec{\delta_k}$	Observation functions error vector
R_k	covariance matrix for $\vec{\delta_k}$
$f_k(\vec{x_k}, \vec{u_k})$	evolution function for the system
F_k	Jacobian of $f_k(\vec{x_k}, \vec{u_k})$
H_k	Jacobian of $h_k(\vec{x_k})$

Table B.1: Definitions of quantities used

References

M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 1999.

F. T. Barath et al. The upper atmosphere research satellite microwave limb sounder instrument. *Journal of Atmospheric Science*, 98:10751–10762, 1993.

P. J. Braspenning, F. Thuijsman, and A. J. M. N. Weijters, editors. *Artificial Neural Networks: An Introduction to ANN Theory and Practice.* Springer-Verlag, 1995.

G. Buskey, G. F. Wyeth, and J. Roberts. Autonomous heleicoter hover using an artificial neural network. In *International Conference on Robotics and Automation*, pages 1635–1640, 2001.

Fredric Chevallier and Jean-Francois Hahfouf. Evaluation of the jacobians of infrared radiation models for variational data assimilation. *Journal of Applied Meteorology*, 40(8):1445–1461, 2001.

Roger Daley, editor. *Atmospheric Data Analysis.* Cambridge University Press, 1991.

Arnt Eliassen. Numerical forecasting. In *Weather Analysis and Forecasting Volume 1*, pages 371–387, 1956.

P. Courtier F. Bouttier. *Data Assimilation Concepts and Methods.* ECMWF Training Course Lecture Notes, 1999.

Scott E. Fahlman. Faster learning variations on back-propagation: An empirical study. In T. J. Sejnowski, G. E. Hinton, and D. S. Touretzky, editors, *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1988.

R. Feltham and G. Xing. Pyramidal neural networking for mammogram tumour pattern recognition. In *IEEE World Conference on Neural Networks*, 1994.

Liang Feng,. 2004. Private Communication.

Mark J. Filipiak. EOS MLS retrieved geophysical parameter precision estimates. Technical Report JPL D-16160, JPL, October 1999. Version 1.1.

L. Garand et al. Radiance and jacobian intercomparison of radiative transfer models applied to hirs and amsu channels. *Journal of Geophysical Research*, 106(D20):24017–24031, 2001.

Ralf Giering. *Tangent linear and Adjoint Model Compiler , Users manual 1.4*, 1999. URL `http://www.autodiff.com/tamc`. Unpublished.

Ralf Giering and Thomas Kaminski. Recipes for Adjoint Code Construction. *ACM Trans. On Math. Software*, 24(4):437–474, 1998.

Simon Haykin. *Neural Networks: A Comprehensive Foundation (Second Edition)*. Prentice-Hall, 1998.

J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.

James P Hollinger, James L Peirce, and Gene A Poe. Ssm/i instrument evaluation. *IEEE Transactions on Geoscience and Remote Sensing*, 28(5):781–790, 1990.

A. K. Hornik, A. M. Stinchcombe, and A. H. White. Multilayer feedforward networks are universal function approximators. *Neural Networks*, 2:359–366, 1989.

Anil K. Jain, Jaingchang Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer Special Edition on Neural Computing*, March 1996.

Micheal A Janssen, editor. *Atmospheric Remote Sensing be Microwave Radiometry*. Wiley-Interscience, 1993.

Carlos Jiménez. *A neural network technique for retrieving atmospheric species from microwave limb sounders*. PhD thesis, Chalmers University of Technology, Sweden, 2003.

Vladimir M. Krasnapolsky. Neural network for standard and variational satellite retrievals. Technical Note OMB Contribution No. 148, National Oceanic and Atmospheric Administration, 1997.

V. M. Krasnopolsky and H. Schiller. Some neural network applications in environmental sciences. part i: forward and inverse problems in geophysical remote measurements. *Neural Networks*, 16:321–334, 2003a.

V. M. Krasnopolsky and H. Schiller. Some neural network applications in environmental sciences. part ii: Advancing computational efficiency of environmental numerical models. *Neural Networks*, 16:335–348, 2003b.

Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann Publishers, Inc., 1992.

Nathaniel J. Livesey and Dong L. Wu. EOS MLS retrieval processes algorithm teoretical basis. Technical Report JPL D-16159, JPL, October 1999. Version 1.1.

Nathaniel J. Livesey et al. Retrieval algorithms for the EOS microwave limb sounder (MLS). *IEEE Transactions on Geoscience and Remote Sensing*, 44 (5):1144–1155, 2006.

A. C. Lorenc. Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 112(474):1177–1194, 1986.

David J C MacKay. Probable networks and plausable predictions - a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, August 1995.

W. S. McCulluch and W. H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133, 1943.

M. Minsky and S. Papert. *Perceptrons: An introduction to Computational Geometry.* MIT Press, 1969.

D. Murtagh et al. An overview of the odin atmospheric mission. *Canadian Journal of Physics*, 80:309–319, 2002.

N. K. Nichols. Data assimilation: Aims and basic concepts. In *Proceedings of the NATO Advanced Study Institute on Data Assimilation for the Earth System*, 2002.

Herbert M. Pickett. Microwave limb sounder THz module on aura. *IEEE Transactions on Geoscience and Remote Sensing*, 44(5):1122–1130, 2006.

Hugh C. Pumphrey. The hydrostatic equation and MLS. hcp@met.ed.ac.uk, 1999.

Hugh C. Pumphrey, 2006. Private Communication.

William G. Read, Zvi Shippony, and W. Van Snyder. EOS MLS forward model algorithm theoretical basis document. Technical Report JPL D-18130, JPL, July 2004. Version 1.0.

B. D. Ripley. Can statistical theory help us use neural networks better? In *29th Symposium on the Interface: Computing Science and Statistics*, 1997.

Clive D. Rodgers. *Inverse Methods for Atmospheric Sounding: Theory and practise (ISBN 981-02-2740-X)*. World Scientific, 2000. ISBN-N 981-02-2740-X.

Raul Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, 1996.

Steve G. Romaniuk. Pruning divide and conquer networks. *Network: Computation in Neural Systems*, 4(4):481–494, November 1993.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

D. E. Rummelhart, G. E. Hilton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rummelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.

Ingrid F. Russell. Neural networks. *Journal of Undergraduate Mathematics and its Applications*, 14(1), 1993.

A. Sakar and R.J. Mammone. Growing and pruning neural tree networks. *IEEE Transactions on Computers*, 42(3):291–299, 1993.

Warren S. Sarle et al. *Usenet Neural Network FAQ.* ftp://ftp.sas.com/pub/neural/FAQ.html, 1997.

Mark R. Schoeberl et al. Overview of the EOS Aura mission. *IEEE Transactions on Geoscience and Remote Sensing*, 44(5):1066–1074, 2006.

Joe W. Waters. An overview of the EOS MLS experiment. Technical Report JPL D-15745, JPL, October 1999. Version 1.1.

Joe W. Waters et al. The earth observing system microwave limb sounder (EOS MLS) on the Aura satellite. *IEEE Transactions on Geoscience and Remote Sensing*, 44(5):1075–1092, 2006.

B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78, 1990.

Julian M. Wright Jr. *Federal Meteorological Handbook No. 3 - Rawinsonde and Pibal Observations.* http://www.ofcm.gov/fmh3/text/default.htm, 1997.

Appendix C

Papers

The following paper has been accepted by the *IEEE Transactions on Geoscience and Remote Sensing* and is currently awaiting publication.

Retrieval of Tangent Pressures from EOS-MLS Radiances Using a Neural Network for use in an Assimilation Scheme

Donald J. Scorgie, Robert S. Harwood, and Hugh C. Pumphrey

Abstract— **Limb sounding instruments provide high vertical resolution data on the temperature and composition of the atmosphere. Their data is therefore valuable for assimilating into general circulation models of the atmosphere. Direct assimilation of radiances from limb sounders is more complex in practise than from nadir sounders due to the need to know the tangent pressures of the measurements. This paper discusses the practical implications of tangent pressures in direct radiance assimilation of limb sounding radiances and demonstrates that a neural network can be used to find these tangent pressures for the EOS-MLS with an RMS error of $\sigma = 50$m, which is comparable with that in traditional retrieval techniques.**

Index Terms— **Microwave, Limb, Neural Network, Tangent Pressure.**

I. Introduction

INDIRECT, or profile, assimilation uses retrieved profiles, such as temperature, from instruments to improve the model's state vector. While this works well for certain types of instrument that measure atmospheric properties directly (e.g. in-situ measurements), for satellite data this introduces several problems [1]. Typically, satellite retrieval systems use an a-priori profile and associated covariance matrix to perform an optimal estimation consistent with the radiances (e.g. [2]), which will result in traces of the a-priori still being present in the final profile. For an assimilation scheme, the a-priori profile is typically unlike either the retrieved profile or the background state of the model. When assimilating, this a-priori may drag the model state away from both the background state and the observations.

Direct, or radiance, assimilation reduces this problem by using the measured radiances directly, thus effectively performing the retrieval as the assimilation step, with the background model state acting as the a-priori. This results in a final state that is a combination of the initial background state and the observations, with no other a-priori.

A problem when attempting direct assimilation of radiances from limb sounding instruments arises from the need to determine pointing information, normally the maximum pressure on the central ray of the field of view, called the *tangent pressure*. This is normally found by a retrieval process and so is not readily available when doing direct assimilation.

This paper investigates whether tangent pressures needed for the assimilation can be provided with sufficient accuracy using a simplified, rapid, limited retrieval scheme based on a neural network, bypassing the need for the complete retrieval step at assimilation time. The method was developed for the EOS-MLS instrument, described in section II. Section III gives details of the neural network adopted and the training procedure. The results and conclusions are given in sections IV and V respectively.

Manuscript received September 11, 2006; revised November 20, 2006. This work was supported by NERC.

II. Measurements from the EOS-MLS instrument

A. Radiances

The Earth Observing System (EOS) Microwave Limb Sounder (MLS) is an instrument aboard the EOS Aura satellite launched in July 2004 [3] [4]. It measures thermal emissions throughout the Earth's limb to determine atmospheric composition and temperature throughout the stratosphere and troposphere.

The EOS-MLS principally uses a band of 25 channels centred on the 118.75 GHz O_2 line to determine temperature and pressure information. The field of view is scanned vertically upward during one scan, producing a series of 125 measurements per scan. Each measurement within a scan is called a *minor frame* and a complete scan, together with ancillary information, is referred to as a *major frame* or *profile*. Figure 1 shows an example of radiance profiles from several channels, where the radiances are expressed as brightness temperatures. The line width is proportional to pressure, so it decreases rapidly as the instrument scans upward through the atmosphere. Hence, the brightness temperature in any one channel will be close to zero above some given point. Below this, the brightness temperature will increase until the atmosphere becomes opaque. Once the atmosphere becomes opaque, the brightness temperature represents the temperature near the height where the opaqueness began.

B. Tangent pressures

As the measurements are strongly dependent on the pressure of the air in the field of view, a convenient vertical co-ordinate is the logarithm of the pressure, $\zeta = -\log_{10}(p/1\text{hPa})$. Retrievals are frequently carried out using ζ as a "height coordinate" and direct assimilation of brightness temperatures often uses the same coordinate.

While the frequency range for a given channel remains the same from one scan to the next, the tangent pressure for each scan step changes across scans, due to spacecraft movement and atmospheric variations. The instrument system has no direct way of measuring tangent pressures although it does

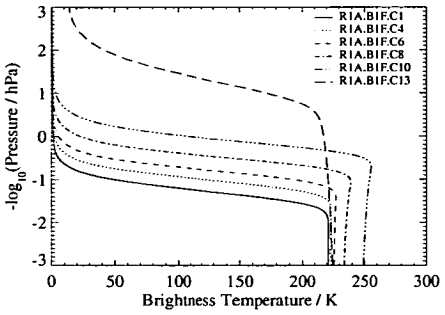

Fig. 1. Simulated radiance measurements from the EOS-MLS. The channels shown here are from band 1 of the instrument which operates near the 118GHz Oxygen line. There are 125 measurements per profile.

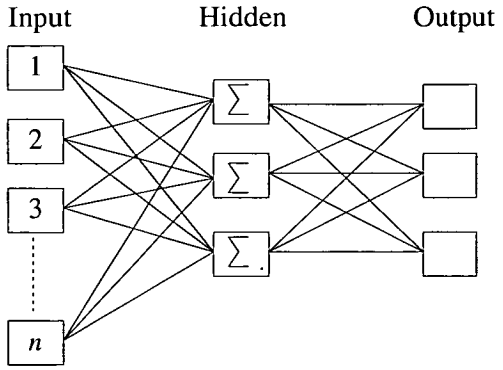

Fig. 2. An example neural network structure with n input nodes, 3 hidden nodes and 3 output nodes.

provide an estimate of geometric height of the tangent point, known as a *tangent height*.

As the line width is a strong function of pressure, the radiances contain information about the tangent pressures. This pressure information is sufficiently decoupled from the temperature information that tangent pressures can be included in the state vector and retrieved, together with the temperatures.

In a direct assimilation scheme, the retrieved products may not be available. Incorporating the tangent pressures into the model state vector is technically complex and outside the scope of this paper [5] and it has been shown that it cannot be assumed the tangent pressures at the same minor frame is the same across profiles [6]. Therefore, a different method must be used to establish the tangent pressures.

III. Neural Networks

Neural networks are used for many purposes, both within and outwith remote sensing [7] [8]. A neural network can be considered as a non-linear fitting technique. The inputs and outputs can be represented as a pair of vectors and the algorithm uses one or more intermediate vectors at so-called "hidden layers". Each element of this intermediate vector is associated with a "node" at which ancillary information, namely a set of "weights", is combined with the inputs in the calculation. These weights are adjusted in a training process to give acceptable results for a set of input-output vector pairs found by other methods.

A graphical representation of a sample neural network is shown in figure 2. Here, there are n inputs, 3 hidden nodes and 3 outputs. The neural network is run by setting the input node values. These are then multiplied by the input-to-hidden weights and passed to the hidden nodes. Here, the inputs to the hidden nodes are summed and an activation is performed and the resulting values are used as the output of the hidden nodes. This activation is typically based on a sigmoid function, given by equation 1, where ω_i are the weights leading into the node and I_i are the corresponding input values. The outputs of the hidden nodes are then multiplied by the hidden-to-output weights and passed to the output layer. Here, each output node sums its inputs and again performs an activation, resulting in a (normalised) output value.

$$ac(\sigma) = \frac{1}{1+\exp(-\sigma)} \qquad (1)$$
$$\sigma = \sum_i \omega_i I_i$$

Training is done here using a three-stage process. The first stage (stage I) runs an input profile from the training set through the network and produces an output. This is then compared to the expected output. The second stage (stage II) involves updating the weights within the system to bring the output closer to the expected output. A second set of profiles, the *validation set*, is then used to assess the suitability of the network (stage III). This validation set is run through the network at regular intervals, and the error calculated. If the error in these validation set is lower than previous errors, the network state is stored. Once the error on the validation set has not improved for a set number of training-validation cycles (epochs), the training of the network is stopped, the best network state is restored and the network is ready for use.

Here, the neural network was trained using the backprop algorithm [9]. This is one of the simplest forms of training algorithm available for neural networks but produces reliable results. Backprop works by calculating a "sensitivity" factor for each node in the network. The weights for that node are then updated using the (first) derivative of the activation function and this sensitivity factor, combined with a "learning rate" and "momentum" that are user-defined. Further details of the algorithm can be found in e.g. [10].

A. Training Data

All the training data were generated by an accurate, full forward model created by H. Pumphrey. The atmospheric temperature and pressure profiles used in generating the radiances were taken from Met Office assimilation data [11] [12], and represent typical conditions encountered by the EOS-MLS. 3496 radiance profiles for band 1 and band 32 of the instrument were then generated from these atmospheric profiles, representing a complete day of measurements by the EOS-MLS.

An alternative, considered but not possible before launch when the bulk of this work was undertaken, would be to use

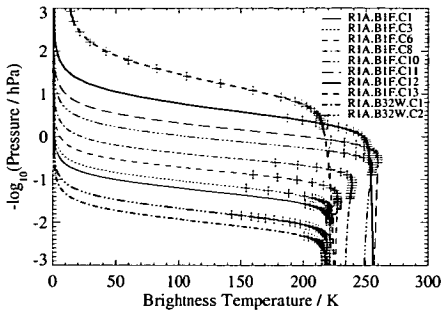

Fig. 3. Example profile from the training set showing the channels and minor profiles used as inputs to the neural network. The crosses represent minor frames that are used. These are chosen as they give the most information about tangent pressures at these heights. There is some overlap between the chosen minor frames. This is done to ensure good results in the network and introduce redundancy, allowing the best fit possible.

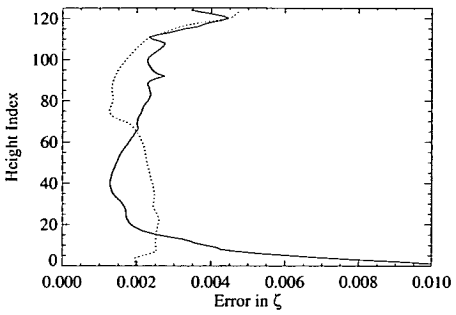

Fig. 4. RMS error for 1000 testing profiles, with no noise associated, run through a fully trained neural network. The largest errors are found at the bottom of the profile, where little information is gathered by the instrument (see figure 3). The dotted line shows the approximate error associated with optimal-estimation retrieval techniques. Between height indexes $15 < h < 112$ (where the instrument gathers information), the RMS error is between $\sigma = 0.0015$ and $\sigma = 0.003$ with a bias of $b < 0.0001$.

the actual MLS radiances with optimally retrieved profiles as the training data. This alternative was rejected however, as in that case the actual values of the "truth" would be unknown and the extra representativeness compared with the present training set is believed to be marginal. Moreover the error covariance matrix (needed in the assimilation) resulting from the MLS-trained network would inevitably be overestimated.

The network was constructed using 200 inputs and 125 outputs. The inputs consist of radiances from different minor frames across several channels in band 1 and band 32 of the instrument. Details of which minor frames are used is given in figure 3. These input radiances provide tangent pressure information from almost ground level to the top of the atmosphere and allow the neural network to retrieve tangent pressures throughout. The outputs of the neural network consists of 125 tangent pressures, one for each scan step within the profile.

B. Training Procedure

The training data were split into three sets, A) 1500 profiles that were used as a training set, B) 300 profiles, used for validation. A final dataset of 1000 profiles (C) was used as a testing set after the training cycle was finished to ensure the network was accurate. The training consisted of the following steps

1 Select one profile from the training set (A) at random (stage I)
2 Train the network with this profile (stage II)
3 Repeat steps 1 and 2 5000 times (profile is randomly chosen each time)
4 Validate the network using all profiles in the validation set (B) (stage III)
5 If the new validation error is less than the current validation error, save the network state
6 After 100 validation runs produce no better error, stop training and restore weights to their best values

Once these steps were completed, the network was tested using the testing set (set C). Using traditional retrieval methods, tangent pressures have an RMS error of ~ 50m (derived from [13]), equivalent to an error in ζ of $\zeta = 0.003$. To be useful, the tangent pressures retrieved using a neural network should have comparable or better errors.

IV. Results

The number of hidden nodes in the network was varied across training runs and ranged from 0 to 50 hidden nodes. It was found that the best results were obtained when using 20 hidden nodes. More than 20 hidden nodes resulted in extra running time with no improvement in error in either the noiseless and noisy case (see below). The use of tangent heights, as inputs to the neural network retrieval, was also investigated but found to produce no effect on the retrieved tangent pressures.

Figure 4 shows an example of a testing run on a fully trained network with 20 hidden nodes. Here, RMS error across the test set for each network output (minor frame) is plotted. The largest errors occur at the bottom of the profile (height index $0 < h < 15$). Above this, the error drops dramatically and again increases slightly at the top of the profile (height index $112 < h < 125$). Between these extremes (height index $15 < h < 112$), the RMS error is $\sigma < 0.003$, in line with the error from traditional retrieval techniques given above.

Below the 15th height index, the errors on network outputs increase to $\sigma = 0.01$, or ~ 180m. These levels correspond to $-3.2 \leq \zeta \leq -2.8$, very near the Earth's surface. At these heights, all the channels of the EOS-MLS instrument are saturated (figure 3). Similarly, above the 112th network output (around $\zeta \geq 1.8$), the atmosphere is thin and all channels register near-background radiation.

The previous results were gathered using noiseless radiances. In practice, the noise associated with the measurements in the channels used here has a standard deviation of $\sigma \approx 0.4$K. To deal with this, a new network was trained in the same way as previously but with all inputs having a randomly generated normally distributed noise associated (that was regenerated every time each profile was used), with a standard deviation of $\sigma = 0.4$K[1]. During the testing phase,

[1]0.4K is used as an approximation across all channels here

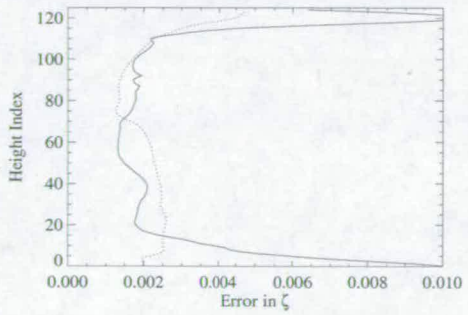

Fig. 5. Results of the testing phase with noise added to the inputs in all data sets. Between height indexes $15 < h < 112$, the RMS error ranges from $\sigma = 0.0017$ to $\sigma = 0.0026$ and a bias of $b < 0.0001$.

noise was again added to each input. The results of a test run using this configuration can be found in figure 5, in the same form as previously. Here, the largest RMS error, between height index $15 < h < 112$, is $\sigma = 0.0026$, which is again comparable to the error achieved using traditional retrieval techniques. In this case, the spike near height index $h = 92$ has been reduced. As different training runs in a neural network produce slightly different results, this is attributed to natural variation between training runs.

V. Conclusion

This paper has addressed the problem of estimating tangent pressures for a limb-sounding instrument for use in a direct assimilation scheme, using a neural network retrieval. It has been shown that this approach can achieve comparable errors to traditional retrieval techniques.

As the tangent pressures are not part of the assimilation model's state vector, the errors associated with their retrieval must be accounted for in the forward model error matrix. This is independent of the method used in retrieving tangent pressures and must be faced however they are retrieved. In a neural network retrieval, the testing phase of the neural network training provides enough information to construct an error covariance matrix.

To use this technique in an assimilation scheme, the tangent pressures would be retrieved prior to the assimilation process and then assumed to be constant within the assimilation, with the error from tangent pressure retrieval considered as part of the forward model error covariance matrix.

Neural network retrieval of tangent pressures in this case provides several advantages over traditional retrieval techniques. They introduce no a-priori estimate of the tangent pressures and are significantly faster. The a-priori is used by traditional retrievals as a starting point for an iterative descent and the final result will always have a component of the a-priori in it.

Acknowledgement

The authors would like to thank NERC for providing the funding for this research and Liang Feng for assistance during this research.

References

[1] P. C. F. Bouttier, *Data Assimilation Concepts and Methods*. ECMWF Training Course Lecture Notes, 1999.

[2] C. D. Rodgers, "Retrieval of atmospheric temperature and composition from remote measurements of thermal radiation," *Reviews of Geophysics and Space Physics*, vol. 14, no. 4, pp. 609–624, 1976.

[3] M. R. Schoeberl *et al.*, "Overview of the EOS Aura mission," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 5, pp. 1066–1074, 2006.

[4] J. W. Waters *et al.*, "The earth observing system microwave limb sounder (EOS MLS) on the Aura satellite," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 5, pp. 1075–1092, 2006.

[5] L. Feng, 2004, private Communication.

[6] D. Scorgie, "A fast forward model for the assimilation of radiances from the EOS-MLS," Ph.D. dissertation, University of Edinburgh, 2006.

[7] C. Jiménez, "A neural network technique for retrieving atmospheric species from microwave limb sounders," Ph.D. dissertation, Chalmers University of Technology, Sweden, 2003.

[8] R. Yasdi, "Prediction of road traffic using a neural network approach," *Neural Computing and Applications*, vol. 8, no. 2, pp. 135–142, 1999.

[9] D. E. Rummelhart, G. E. Hilton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rummelhart and J. L. McClelland, Eds., vol. 1. MIT Press, 1986.

[10] P. J. Braspenning, F. Thuijsman, and A. J. M. N. Weijters, Eds., *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. Springer-Verlag, 1995.

[11] G. L. Manney *et al.*, "Comparison of U.K. Meteorological Office and U.S. National Meteorological Center stratospheric analyses during northern and southern winter," *Journal of Geophysical Research*, vol. 101, no. D6, pp. 10 311–10 334, 1996.

[12] R. Swinbank and A. O'Neill, "A Stratosphere-Troposphere data assimilation model," *Mon. Weather Rev.*, vol. 122, pp. 686–702, 1994.

[13] M. J. Filipiak, "EOS MLS retrieved geophysical parameter precision estimates," JPL, Tech. Rep. JPL D-16160, October 1999, version 1.1.

Donald Scorgie Donald Scorgie studied maths and physics at Manchester University, emerging with a M.Math.Phys in 2002. Since then, he's been working towards his Ph.D, entitled "A Fast Forward Model for the Assimilation of Radiances from the EOS-MLS", at the University of Edinburgh.

Robert Harwood "Bob" Harwood is Professor of Atmospheric Science at Edinburgh University. He obtained B.Sc. in mathematics and Ph.D. in Meteorology at Imperial College London, and spent 7 years at Oxford University, moving to Edinburgh in 1976. His research interests are remote sensing and computer simulation of the atmosphere at global scales.

Hugh Pumphrey Hugh Pumphrey received a BA in physics from Cambridge in 1986 and a Ph.D in physics from the University of Mississippi in 1989. After working for three years as a research fellow in underwater acoustics at Cambridge University, he moved to Edinburgh, where he has remained, first as a research fellow, then as a reader. His current research interests are in the field of atmospheric remote sounding. He is a co-investigator on the EOS MLS instrument.