Implementation of a Neural Network for Retrieving Atmospheric Parameters from Remote Sensing

Philippe BARON, Jana MENDROK, and KASAI Yasuko

A numerical model of a supervised feedforward Neural Network (NN) has been implemented. The purpose is to study the capabilities of a NN based retrieval algorithm to inverse the measurements performed by the future JEM/SMILES limb sounder. The model has been designed for research purpose with a special care given to its flexibility and extension facility, but keeping in mind that the computational performances must allow the use of a network with the size of those commonly used for satellite data inversion. The code is written in the Python language. The procedure to create and use a NN is presented and the algorithms of the training procedure are described in detail. The MLP is trained using either the Levenberg-Marquardt or the steepest descent method to find the optimal value of the model parameters according to some examples of the inputs and outputs. The model also provides a set of functions to scale the data or to use their principal components. In order to prevent the MLP to overfit the training data, several solutions are available. A regularization term can be added to the cost function with the possibility to optimize the hyperparameters using a Bayesian method. Also, an early stopping procedure can be set using a cross-validation data set. The correctness of the algorithms implementation is demonstrated with simulations and the results are discussed.

Keywords
Neural network, Multilayer Perceptron, Supervised learning, Satellite remote sensing, Retrieval

1 Motivation

The super-conductive Sub-Millimeter Limb Emission Sounder (SMILES) is a high sensitive radiometer planned to be launched in 2009 in order to study the chemistry of the mid-atmosphere (from ~10 km to 80 km)[1]. It will operate from the Japanese Experiment Module (JEM) aboard the International Space Station to measure trace gases on a global scale. A data processing chain, based on the Optimal Estimation Method (OEM)[2], is under development by the Japanese Aerospace eXploration Agency (JAXA). The procedure uses a time consuming iterative procedure in order to inverse the measurements using a non linear forward model with respect to the retrieved parameters[3].

An analysis is currently been performed in NICT to study the capability of a Neural Network (NN) based retrieval algorithm to process the JEM/SMILES data. For this purpose a supervised feed-forward NN, i.e., a special family of a NN[4], has been developed. A feed-forward network is made of successive layers of neurons and process the information from the first layer to the last one without feedback. This paper will mainly deal with a MultiLayer Perceptron (MLP)[1], a special case of feedforward network that is com-

monly used for retrieval applications. The term supervised means that the model parameters are optimized during a learning (or training) procedure with respect to a set of examples of the inputs and outputs.

A NN offers an alternative approach to solve a non-linear retrieval problem. The NN output calculation is fast and only requires small memory storage. The computation time cost is paid during the training phase that is performed prior and independently to the data processing phase. The training data can be produced using the forward model and does not required real measurements. Furthermore the weighting functions with respect to the retrieved parameters are not needed.

Theoretical studies for various satellite observations techniques have shown that a MLP based algorithm provides similar performances (e.g., precision, accuracy and vertical resolution) as the traditional statistical methods[5]-[9]. Moreover some implementations have already been successfully applied to real measurements[7][10]-[14]. Among these references, a special attention is paid to the inversion of the measurements performed by the Sub-Millimeter Radiometer (SMR) aboard the Odin satellite[15]. The JEM/SMILES measurements will be similar to the Odin/SMR ones and the retrieval procedure must deal with equivalent problems.

This paper focus on the model itself and the application to JEM/SMILES data processing is beyond its scope. It is worthwhile to note that the model application is not only limited to the JEM/SMILES retrieval analysis but can also be used to solve any non-linear regression problems or more generally to solve function approximation and classification problems[16].

Section **2** presents the general features of the model and how to create and use it. The model implementation is also discussed. Section **3** details the algorithms of the learning procedure and section **4** gives the different ways in the model to avoid the common overfitting problem when the network is trained. Finally the validity of the algorithms implementation is demonstrated using simulations in Section **5**.

1 A MLP is made of at least one hidden-layer. The hidden neurons perform a derivable non-linear function of their inputs and the output neurons perform a linear operation. Its has been demonstrated that a MLP can approximate any regular function with a given precision

2 General features

The trained MLP can approximate the unknown inverse function of the JEM/SMILES retrieval problem:

$$y_n \approx \mathrm{MLP}\,(x_m, w_p) \tag{1}$$

where y_n is a n-dimension vector that represents the unknown atmospheric states as well as unknown instrumental parameters, x_m is a m-dimensional vector that represents the noisy measurements, and w_p is p-dimension vector of the MLP parameters[2].

A MLP works as an interpolator of any regular function defined on a sampled grid of q elements, i.e., the learning data set $\{x_m, y_n\}_q$. This analogy makes obvious the limitations of the model: i) a wrong result outside the m-dimension space covered by the training set, ii) poor performance possible for sparse training data, iii) sensitivity to the noise/error on the training data.

For the JEM/SMILES retrieval analysis, the forward model will be used to produce the training data set including the knowledge of the statistic of both the instrumental errors and the atmospheric variability. It should be noticed it is also be possible to use real measurements instead[12].

As shown in Figure 1, the MLP is composed of neurons organized in successive layers. The first and the last layers are called the input and output layers, respectively. They both enclose the hidden layers. The neurons of the input layer does not perform any operation but just present the inputs to the neurons of the first hidden layer. The neurons of the other layers perform an operation of the form:

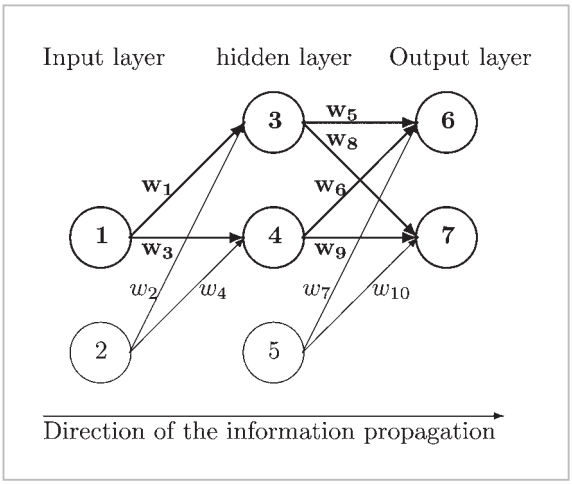

Fig. 1 *Sketch of a 2 layers network with 1×2×2 neurons (i.e., 1 input neuron, 2 output neurons and one hidden layer of 2 neurons) and biases*

A bias is represented as the weight of the connection between a virtual neuron with a constant activation 1 and a real neuron. The real neurons are represented with the thick circle 1,3,4,6,7 and the biases with the thin circles 2 and 5. The weights of the connection w_1 to w_{10} are the adjustable parameters of the model.

$$a[j] = f\left(\sum_i w_{ji}\, a[i] + \theta[j]\right) \tag{2}$$

where $a[i]$ and $a[j]$ are the activation levels of the neurons i and j, the index i runs over all the neurons located in the layer just before the layer containing the neuron j, w_{ij} is the weight of the connection between the neurons i and j and $\theta[j]$ is the bias. The weights w and the biases θ are the free parameters of the model. The activation of the neurons of the last layer are the MLP outputs.

By default $f(x)$ = tanh for the hidden neurons (i.e., neurons in the hidden layers) and f is the identity function for the output neurons. The user can defined its own activation function with the constraint that it is assigned to all the neurons in a same layer.

In this paper, the size of the network will be defined by the number of hidden layers plus the output layer. The number of neurons will be indicated using the notation, $Ni \times N1 \times N2 \ldots \times No$, where Ni is the number of neurons in the input layer, $N1$ and $N2$ the number of neurons in the first and second hidden layers, and No the number of neurons in the output layer. As example, Fig. 1 shows a 2 layers network with $1 \times 2 \times 2$ neurons. It should be noted that the biases are represented as virtual neurons with a constant activation value but are not taken into account when indicating the neurons.

2 in this paper the underscripts indicate the dimension of the vector/matrix and the superscripts T and -1 indicate the matrix transpose and inverse operators, respectively

2.1 Data normalization and compression

The quality and the robustness of the learning procedure can be improved when the learning algorithm is applied to scaled and compressed data[7][8][17]. The model provides the possibility to perform these transformations. When these options are switch on, they are hidden for the users and the model inputs and outputs remain the same.

The scaled input data $\bar{x}_m$ are calculated as:

$$\bar{x}_m = \frac{x_m - \text{mean}(\tilde{x})_m}{3 \times \text{std}(\tilde{x})_m} \tag{3}$$

where $\tilde{x}$ is a representative sample of the inputs x_m.

The data compression is based on a Principal Component Analysis (PCA). The compressed input κ_a is the representation of x_m on the basis composed by the first a eigenfunctions of the covariance matrix of a representative sample $\tilde{x}$:

$$\kappa_a = U_{a,m} x_a . \tag{4}$$

The rows of the matrix $U_{a,m}$ are the a first orthonormal eigenfunctions sorted by decreasing eigenvalues (i.e., the variance of the components).

The compression level (i.e., the ratio a/m) is controlled by the error on the reconstructed input x_n^r when the inverse transformation is realized:

$$x_m^r = \left(U^T\right)_{m,a} \kappa_a . \tag{5}$$

The user defines the threshold under which the sum of the squared error $\sum_{i=1,m}(x^r[i]-x[i])^2$ should lie.

The method is based on the assumption that the principal components with low variances (small eigenvalue) carry poor informations and can be neglected. It means that the information about parameters with low variability signature in the measurement might be rejected by the compression procedure. The low signature in the measurement can be due to the low variability of the atmospheric parameters in the training set or to the low sensitivity of the measurement.

It should be noted that although we only described the normalization and PC transformations on the inputs, the same operations can also be applied to the outputs y_n.

2.2 Implementation description

The model has been designed for research purpose with emphasis on its flexibility and extension facility. However the model must be able to handle MLP with the size of the ones commonly used for satellite measurements inversion. As example, a 30×10×1 MLP and a training set of 4000 examples is used to invert the Odin/SMR measurements[11] and retrieve one atmospheric parameter per network. In the analysis in the reference[18] the temperature profile is retrieved from the simulated Infrared Atmospheric Sounding Interferometer (IASI) measurements using a 671×50×30 MLP and a training set of 2700 examples. In the Odin/SMR, the learning procedure has to infer the optimal values for 321 parameters from 4000 equations; but in the case of IASI, the system increases to 35130 parameters and 81000 equations.

Given these constraints, the code has been implemented in the interpreted object-oriented Python programming language (http://www.python.org/). The code itself is composed of 3 files and uses the external library NumPy (http://www.scipy.org/) to provide the general mathematical functions as well as the capability to manipulate arrays and, so, limit the use of the slow loops that are one drawback of the interpreted language. Let's note that both numpy and the 2-D plotting library matplotlib (http://matplotlib.sourceforge.net/) transforms Python in a practical and complete environment for this analysis. The model has been tested on both Linux and Windows operating systems. So far, the CPU time required to train the Odin/SMR network on a normal PC is few minutes and several days to train the IASI one. We can consider that the later case is the limitation of the model. However the code implementation will still be improved in the future and meanwhile, the performance is good enough to perform JEM/SMILES retrieval.

The MLP is a Python object instance. It provides a set of methods (i.e., functions) to modify and access its attributes (i.e., properties) such as the topology, the weights or the activation level of each neuron. The different operations to set up and run the model can be written in a Python script and executed as a normal program. They can also be directly executed from the Python shell. In this case, the MLP execution is performed in an interactive way and the MLP properties can be analyzed after each operation.

As example, Table 1 shows some commands in order to approximate a function $Y_2 = f(X_4)$ with a 4×3×2 MLP. It should be noted that list of the methods that are presented is not exhaustive.

In this example, first the MLP module is loaded as the python module import (step 1). The 4×3×2 MLP with biases is created as an

Table 1 *An example of commands*

The sequence of the operations to approximate a function $Y_2 = f(X_4)$ using a 4×3×2 MLP and training data set $\{Xt_{4,q}, Yt_{2,q}\}$ of q examples. The meaning of the steps is described in the text.

```
1> import mmlp
2> m = mmlp.Mlp([4,3,2], bias=1,
     act_hidden='tanh', act_last='linear')
3> m.setPcx(Xt, 1.e-6)
4> m.setScaley(Yt)
5> m.initweights()
6> m.fitweights(Xt,Yt)
7> m.run(X)
8> Y = m.getY()
```

object named m (step 2). It is set with the tanh and linear activation functions for the hidden neurons and the output neurons, respectively. The input data are transformed to the PC of the sample $Xt_{4,q}$ of q examples (step 3), and the output data are scaled with respect to the sample $Yt_{2,q}$ (step 4). The threshold controlling the compression level of the output data is set to 10^{-6}. The weights are randomly initialized (step 5) and computed according to the training data set $\{Xt_4,\ Yt_2\}_q$. Finally, the model is used to process the input X_4 (step 7) and the output Y_2 is retrieved (step 8).

3 The learning algorithms

3.1 The regression algorithm

The training procedure aimed to estimate the optimal values of the MLP parameters w_p using the training data $\{x_m, z_n\}_q$ of q inputs x_m and expected values z_n. The solution minimizes the cost function ε^2 defined as:

$$\begin{aligned}\epsilon^2 &= \frac{\beta}{2}\left[\tilde{z}_o - \tilde{y}_o(\{x_m\}_q\ ,\ w_p)\right]^T\left[\tilde{z}_o - \tilde{y}_o(\{x_m\}_q\ ,\ w_p)\right] \\ &+ \frac{\alpha}{2} w_p^T w_p \qquad (6)\\ &= \beta E_y^2 + \alpha E_w^2\end{aligned}$$

where $o = n \times q$, $\tilde{z}_o$ is the vector built with the set of expected values $\{z_n\}_q$, and $\tilde{y}_o$ is the outputs when all the training inputs $\{x_m\}_q$ are presented to the model. The term E_w is included in order to stabilize the solution and to favour low values of w.

The minimum of the cost function is located using the Steepest (or gradient) Descent (SD)[2][4] or the Levenberg-Marquardt (LM)[2][19] methods. Both methods are iterative algorithms that correct the weights after each iteration as:

$$dw_p = -\eta\left(\frac{\partial \epsilon^2}{\partial w_p}\right)_p . \qquad (7)$$

The derivative of the cost function is given by:

$$\left(\frac{\partial \epsilon^2}{\partial w_p}\right)_p = -\beta\, \mathrm{J}_{o,p}^T\left[\tilde{z}_o - \tilde{y}_o(\{x_m\}_{1\ldots q}\ ,\ w_p)\right] + \alpha\, w_p \qquad (8)$$

where $\mathrm{J}_{o,p} = \frac{\partial \tilde{y}_o}{\partial w_p}$ is calculated using the standard retropropagation algorithm[4] and the parameter η is:

$$\eta = \begin{cases} \mu^{-1}\ , & \text{in case of SD} \\ \left[\beta(\mathrm{J}^{\mathrm{T}}\mathrm{J})_{p,p} + (\mu+\alpha)\,\mathrm{I}_{p,p}\right]^{-1}, & \text{in case of LM}\end{cases}$$

where $\mathrm{I}_{p,p}$ is the $p \times p$ identity matrix, and μ is the inverse of the step size parameter for the SD method and the Marquardt parameter for the LM method. The parameter η for LM is derived from the Gauss approximation of Hessian matrix of the cost function:

$$\frac{\partial^2 \epsilon^2}{\partial w_p^2} \approx \left[\beta(\mathrm{J}^{\mathrm{T}}\mathrm{J})_{p,p} + \alpha\,\mathrm{I}_{p,p}\right]. \qquad (9)$$

The initial value of μ is set by the user (the default value is 10) and it is updated after each iteration in order to ensure the convergence toward the cost function minimum. The Marquardt strategy for updating μ is used for both the SD and LM methods. If the cost function (Eq 6) increases after updating w_q, the parameter μ is multiplied by an user defined value (the default value is 10) and a new estimation of the weights is performed (Eq 7). If the cost function decreases, μ is divided by an other user defined value, the Jacobian and the Hessian matrices are calculated and w_q is updated. Such process is repeated until the convergence is reached. In case of LM, once the convergence is reached an additional loop can be performed with μ set to 0.

The convergence is reached when the change of the cost function between two consecutive iterations i and $i + 1$ is small. The convergence criterion is defined as $|\varepsilon^2\ (i + 1) - \varepsilon^2\ (i)| < (\mathrm{thr} \times \varepsilon^2\ (i))$. The threshold thr is defined by the user (the default value is 10^{-5}). We present in section 4 some additional mechanisms that have been implemented to stop the iterative process before the convergence is reached.

At the beginning of the iterative process, the biases are set to 0 and the connection weights are randomly set with value between −1 and 1. When the input data are normalized, it is also possible to use the method proposed by[20] to initialized the weights and the biases between the input and the first hidden layers.

The LM method is stable and is faster to converge to the solution than the SD method. However SD does not required the calculation and the inversion of the Hessian matrix (Eq 9) and, so, it is better suited to train a network with a large number of parameters such as the IASI one presented in Section **2.2**.

The matrix $J_{o,p}$ (Eq 8) has a large size but it is not stored in memory since only the Hessian matrix (Eq 9) and the Jacobian of the cost function (Eq 8) are used. In the model the training set is decomposed into n subsets of q elements, where each subset corresponds to one neuron output. The Hessian and the Jacobian matrices are calculated using the equations:

$$\left(\mathrm{J}^T\mathrm{J}\right)_{p,p} = \sum_{i=1,m} \left(\mathrm{Ji}^{\mathrm{T}}\right)_{p,q} (\mathrm{Ji})_{q,p}$$

and

$$\left(\frac{\partial \epsilon^2}{\partial w_p}\right)_p = -\beta \sum_{i=1,n} \left(\mathrm{Ji}^T\right)_{p,q} \tilde{e}^i{}_q + \alpha w_p$$

where $\mathrm{Ji}_{q,p}$ and $\tilde{e}^i{}_q$ are the Jacobian matrix and the error on the activation of the i^{th} network output estimated on the q training data.

3.2 Bayesian regularization

A probabilistic approach of the regularized regression problem leads to interpret the hyperparameters α and β (Eq 6) as the inverse of the variances of the MLP parameters w_p before the training process, and the error $\hat{E}_y = [\tilde{z}_o - \tilde{y}_o(\{x_m\}_q, \hat{w}_p)]$, respectively. The parameter $\hat{w}_p$ is the most probable value of w_p. The notation $\hat{}$ will be used hereafter to define a parameter estimated with $\hat{w}_p$.

An optimal value of the hyperparameters is estimated applying the Bayesian rules[21][22]:

$$\begin{aligned} \alpha &= \frac{\gamma}{2\hat{E}_w^2} \\ \beta &= \frac{o-\gamma}{2\hat{E}_y^2} \end{aligned} \qquad (10)$$

where $\hat{H}_{p,p}$ is the Hessian matrix of the cost function(Eq 9), and

$$\begin{aligned} \gamma &= p - \alpha \,\mathrm{trace}(\hat{H}_{p,p}^{-1}) \\ &= \sum_{k=1}^{p} \frac{\hat{\lambda}_k}{\hat{\lambda}_k + \alpha} \end{aligned} \qquad (11)$$

is the effective number of parameters that can be calculated from the eigenvalues $\hat{\lambda}_k \text{ of } \beta\left(\hat{J}^T\hat{J}\right)_{p,p}$.

When the LM method is used, an algorithm has been proposed to update the hyperparameters after each successful iteration using the values of w_p and $H_{p,p}$ available at the current stage of the iterative cycle[23]. If the SD method is selected, the Hessian matrix has to be calculated. However, this algorithm may leads in some cases to too high value of (R) and overconstrains the solution. Therefore, two alternative implementations, claimed to be more robust, have been preferred in the model.

The first one[24] estimates the parameters from the iterative process:

$$\begin{aligned} \gamma(i+1) &= \sum_{k=1}^{p} \frac{\lambda_k}{\lambda_k + \alpha(i)} \\ \alpha(i+1) &= \frac{\gamma(i+1)}{2E_w^2} \end{aligned} \qquad (12)$$

where i is the iteration index. The number of loops i_{max} is defined by the user and the default number is set to 1 that is the solution proposed in[23]. The parameter β is derived from Equation 10 using the last estimation γ (i_{max}).

The second implementation[25] approximates α as:

$$\alpha \approx \alpha(0)\frac{p}{p-\gamma(0)}$$

and both γ and β are computed using the new α.

The initial values of 0, 1 and p are used for α, β and γ, respectively. As described in[25], the hyperparameters are updated only after a given number of iterations defined by the user (the default number is 3).

After updating the hyperparameters, the definition of the cost function changes and it must be estimated again before to proceed further in the Marquardt loops cycle. The value of the new cost function becomes equal to the

value of γ (See Eq 6 and 10).

The effective number of parameters γ is the degrees of freedom of the network. It is always inferior to the number of parameters w_p. Hence its optimal value provides an objective criterion to estimate the optimal size of the network. For a network with too few neurons, γ is close to the number of parameters w_p. When the size of the network increases, The value of γ increases up to the optimal degrees of freedom. The best size for the network is the one for which the number of parameters w_p is slightly superior to the maximum value of γ.

The complete teaching procedure is summarized in the Table 2.

Table 2 *Summarize of the main steps in the learning algorithm*

The sign* indicates an optional step

1.	*scale the input/output data (Eq 3)
2.	*convert the input/output to PC (Eq 4)
3a	compute the cost function ϵ^2 (Eq 6)
3b	calculate the Jacobian of ϵ^2 (Eq 8)
3c	*calculate the Hessian matrix (Eq 6)
4.	update the weights and biases w_p (Eq 9)
5.	in case of LM:
5a.	if $\mu \neq 0$ and the convergence is reached, set $\mu_{prev} = \mu$ set $\mu = 0$ and return to step 3b.
5b.	if $\mu = 0$ and the convergence is reached, stop the process
5c.	if $\mu = 0$ and ϵ^2 increases, set $\mu = \mu_{prev}$ and return to step 3a.
6	in case of SD:
6a.	if the convergence is reached, stop the process
7.	if ϵ^2 decreases, increase μ set w_p to the previous value and return to step 4.
8a	*in case of Bayesian regularization, if ϵ^2 increases, update the hyperparameters (Eq 10) and ϵ^2
8b	if ϵ^2 increases, decrease μ and return to step 3b

The convergence is reached if one of the following tests is true:
1. small change of the cost function: $\left|\epsilon^2(i+1) - \epsilon^2(i)\right| < (\text{thr} \times \epsilon^2(i))$,
2. number of iterations superior to a threshold,
3. early stopping criterion fulfilled (see section 4.2)

4 Overfitting issue

Overfitting is a common problem when teaching a MLP with noisy data. The noise patterns on the training data are perfectly represented but the MLP has poor performance when the new input data are presented. Several methods have been implemented in the model to deal with this problem and to force the MLP to provide a smooth representation of the training data.

4.1 Structural stabilization and regularization

The complexity of the MLP, i.e. its ability to represent a function with complex patterns, is controlled by the number of parameters w_p and their amplitudes. The complexity of the model depend on its degrees of freedom that can not be superior to the number of parameters. Using a regularized cost function (Eq 6) or acting on the number of parameters are two similar ways to control the degrees of freedom of the model. It should be noted that a model with too low degrees of freedom will not correctly approximate the function.

To solve a retrieval problem it is enough to use a 2 layers networks[14][18]. As already discussed in Section **3.2**, the Bayesian learning procedure provides the optimal value of the degrees of freedom, γ (the effective number of parameters), and, hence, provides a way to estimate the best number of hidden neurons.

4.2 Early stopping

The iterative process of the learning algorithm can be stopped before it reaches the convergence. The early stopping procedure prevents to get the parameters w_p with too large values. The user can stop the learning procedure by defining the maximum number of iterations. Another way is to use a cross-validation data set that indicates when the generalization capability of the model starts to degrade. The process is stopped when the cross-validation cost function does not decrease for a given number of iterations that is defined by the user.

5 Verification of the algorithms

5.1 Neurons activation and output Jacobian matrix

The algorithms implementation for the calculation of the neurons activation and the output Jacobian have been verified using a reference MLP with known parameters. The reference MLP is a simple 1×2×2 network as the one presented in Fig. 1. The values of the parameters w_p are given in Table 4. The activation functions are the tanh and the identity functions for the hidden and the output neurons, respectively. Figure 2 shows the 2 network outputs between −50 and 50.

The small size of the network allows to easily derived the analytical expression of the neurons activation and the Jacobian of the outputs. The Table 3 shows the functions for the activation of the hidden neuron $a[3]$ and the output neuron $a[6]$. Also shown is the Jacobian functions of the first model output $o[1] \equiv a[6]$ with respect to the parameters $w[1]$, $w[5]$, $w[7]$ and $w[9]$.

The model outputs and the analytical functions have been calculated on a regular grid of 10 points between −50 to 50. The activation of all the neurons has been checked as well as the elements of the output Jacobian matrix. The comparison reveals that the differences between both calculations are negligible with an amplitude at the level of the numerical noise ($\sim 10^{-16}$). The Table 5 shows an example of the results for the activation of the neurons 3,4,6 and 7 and the Jacobian matrix of the output 1. For each parameter, it is also indicated the minimum and the maximum values, as well as the maximum of the error.

Table 3 *Analytical expression of the activation of the neurons a(3) and a(6) of a MLP as the one presented in the Fig. 1 along with the Jacobian of the first output o(1)≡ a(6) with respect to the parameters w(1), w(5), w(7) and w(9)*

A tanh and linear activation functions are performed by the hidden and the output neurons, respectively.

$$
\begin{aligned}
a[3] &= \tanh\left(w[1] \times a[1] + w[2]\right) \\
a[6] &= w[5] \times a[3] + w[6] \times a[4] + w[7]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{J}[1,1] = \frac{\partial o[1]}{\partial w[1]} &= a[1] \times w[5] \times \left(1 - a^2[3]\right) \\
\mathrm{J}[1,5] = \frac{\partial o[1]}{\partial w[5]} &= a[3] \\
\mathrm{J}[1,7] = \frac{\partial o[1]}{\partial w[7]} &= 1 \\
\mathrm{J}[1,9] = \frac{\partial o[1]}{\partial w[9]} &= 0
\end{aligned}
$$

5.2 Cost function minimization

The reference MLP defined in the previous section is used to check the implementa-

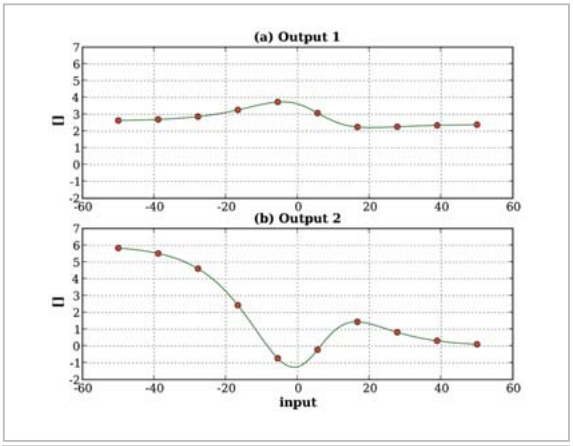

Fig.2 *First and second outputs of the reference MLP*

The red-circle data are used to verify the algorithms.

Table 4 *Values of the reference MLP parameters w_p together with 2 other combinations that produce identical model outputs*

Bias indexes are written with bold Figures.

index	w_p	other solutions	
1	0.05	-0.05	0.10
2	**0.10**	**-0.10**	**-0.50**
3	-0.10	-0.10	0.05
4	**0.50**	**0.50**	**0.10**
5	1.90	-1.90	-2.00
6	2.00	2.00	1.90
7	**2.50**	**2.50**	**2.50**
8	-10.00	10.00	7.00
9	-7.00	-7.00	-10.00
10	**3.00**	**3.00**	**3.00**

tions of the LM and SD methods in the calculation of the model parameters w_p. A new MLP with the same size and activation functions is trained to reproduce the reference model. The training data is made of 10 outputs of the reference MLP calculated with inputs regularly spaced between −50, 50 (see Fig. 2). Three learning setting have been defined: 1) the LM method is used, 2) the LM regression method is used with scaled training data and, 3) the SD regression method is used with scaled data. Table 6 summarized the setting and the results. No regularization term is used in the cost function (β=1 and α=0). The iterative process is stopped when the change of the cost function between 2 consecutive iterations is small, the value depend on the setting (see Table 6). The total number of iterations is limited to 40000. Each setting is repeated 100 times, the initial values of w_p are changed in each run using the Nguyen algorithm (see Section **3.1**).

Table 5 *Comparison of the model with the analytical calculation for 10 inputs regularly spaced between −50 to 50*

The model is the MLP presented in Fig. 1. It is shown the results for the activations $a[3]$, $a[4]$, $a[6]$ and $a[7]$, as well as the first output $o[1] \equiv a[6]$ Jacobian with respect to the 10 weights and biases. The first column gives the variability range of the parameters and the second column gives the maximum error.

	value range	error
Neuron output		
$a[3]$	-0.98, 0.98	1.1e-16
$a[4]$	-1.00, 1.00	1.0e-16
$a[6]$	2.20, 3.74	4.4e-16
$a[7]$	-1.25, 5.83	1.7e-15
Jacobian for $o[1]$		
$\mathbf{J}[1,1]$	-19.29, 14.89	1.5e-14
$\mathbf{J}[1,2]$	0.04, 1.89	4.4e-16
$\mathbf{J}[1,3]$	-4.30, 15.72	9.7e-15
$\mathbf{J}[1,4]$	0.00, 1.99	4.4e-16
$\mathbf{J}[1,5]$	-0.98, 0.98	1.1e-16
$\mathbf{J}[1,6]$	-1.00, 1.00	1.1e-16
$\mathbf{J}[1,7]$	1.	0.
$J[1,8:10]$	0.	0.

It is worthwhile to note that some permutations of the neurons and changes in the sign of w_p let the model outputs unchanged (see Table 4). Therefore a direct comparison of the retrieved parameters with those of the reference model is not well appropriated. The quality of the learning procedure is estimated by the sum of the squared error on the trained outputs deduced from a test data set. The test data are calculated using 100 inputs regularly spaced from −50 to 50.

The top panel of the Fig. 3 shows errors on the trained MLP with the first learning setting. The errors are shown with respect to the run indexes and the inputs of the test set. Among the 100 runs, 88 produced a model with a negligible error of less than 10^{-15}. We can note that the other 12 runs have produced a wrong MLP with a large error that above 0.5 for all inputs. Such wrong model are clearly detected and can not be misinterpreted.

The bottom panel of the Fig. 3 shows the variations of the cost function (calculating on the training data set) and the Marquardt parameters with respect to the iteration index for 5 typical runs. For 4 runs, the convergence is

Table 6 *Definition and results summarized for the three settings*

The results are the average value over the 100 runs performed for each setting. It is given the cost function at the end of the training procedure. The runs with a costfunction > 0.01 are flagged incorrect. The number of successful iterations (decrease of the cost function) and the ratio of successful iteration. The latter is as the ratio of the number of successful iterations to the total number.

Setting	1	2	3
regression method	LM	LM	SD
X and Y Scaling	no	yes	yes
number of runs	100	100	100
convergence threshold	1.e-7	1.e-7	1.e-8
μ initial	100	10.	50.
μ increase rate	2.	2.	150
μ decrease rate	10.	10.	300.
Results			
cost function	4.0e-29	3.7e-29	6.5e-05
number of incorrect runs	12	0	0
number of iterations	25	13	18704
successful iteration ratio	0.36	0.67	0.47

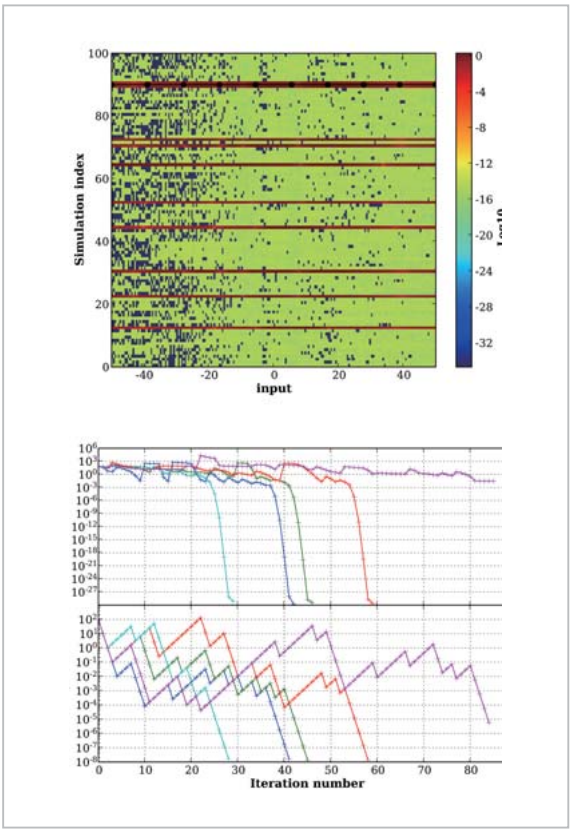

Fig.3 *Results of the first learning setting*

Top panel: error on the trained MLP outputs. The error is defined as the square root of the sum of the squared errors on both outputs estimated for each test data. The Levenberg-Marquardt method is used and the training data are not scaled. The position of the training data are indicated by the full dark circles. Bottom panel: Cost function estimated on the training data set and μ parameter with respect to the iteration index for 5 networks among the 100 runs.

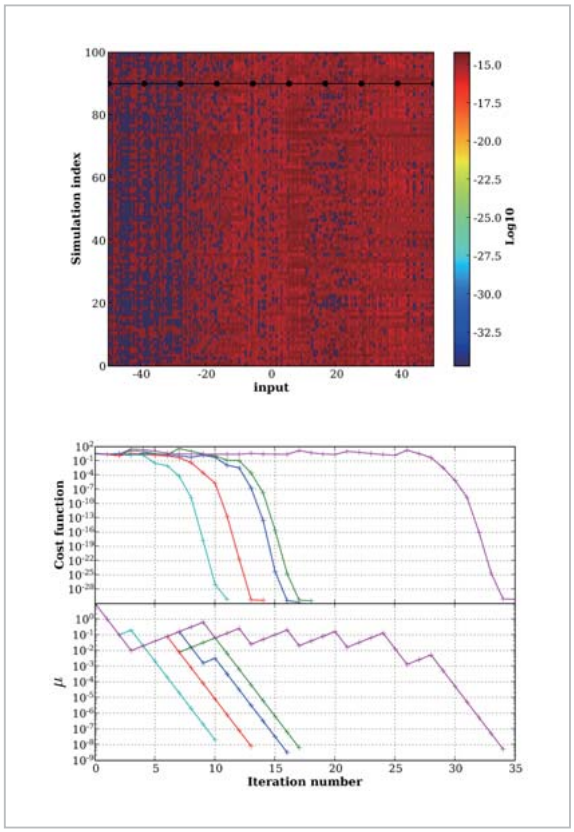

Fig.4 *Results for the second learning setting*

Top panel: Same as the top panel of Fig. 3 but using scaled training data. Bottom panel: Same as the bottom panel of Fig. 3 but using scaled training data.

reached with a low value of the cost function of about 10^{-27} and with 30 to 60 iterations. The $5th$ case is one of the 12 wrong trained MLP, it is trapped in a secondary minimum with a cost function above 10^{-2}.

The Fig. 4 shows the results for the second learning setting. The fact to scale the data improved the results since all the trained MLP have reproduced perfectly the reference model with a error less 10^{-14}. The convergence is always reached with less than 35 iterations. The mean number of successful iterations is 13.

The results obtained from the two first settings demonstrate the validity of the Levenberg-Marquardt algorithm implementation.

The results for the last learning setting are shown in Fig. 5. The training is done using the SD method on scaled data. The convergence is very slow and, for 99 runs, the training has been stopped before the convergence is reached. The mean number of successful iterations is about 18000. For the run represented by the horizontal dark red line around the simulation index number 10, the learning procedure has incorrectly interpreted the small change of the cost function as a convergence and stopped the process too early. However the convergence is stable since all the trained MLP has reproduced the error patterns with an error inferior to 10^{-2}. We can conclude that the steepest descent algorithm is correctly implemented but the convergence speed has to be increased by improving the update of the step size parameter (μ^{-1}) after each iteration.

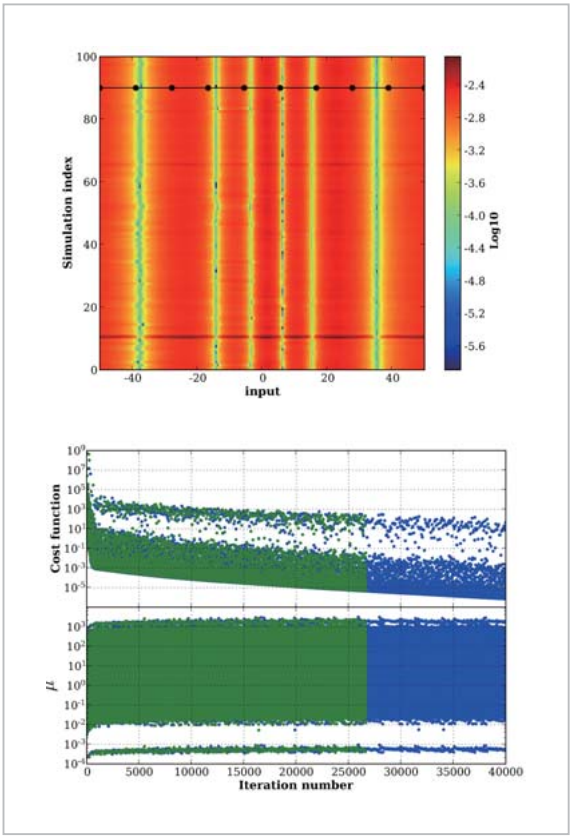

Fig.5 *Results for the last learning setting*

Same as the top panel of Fig. 3 but using the steepest descent method and scaled training data. Bottom panel: Same as the bottom panel of Fig. 3 but using the steepest descent method and scaled training data.

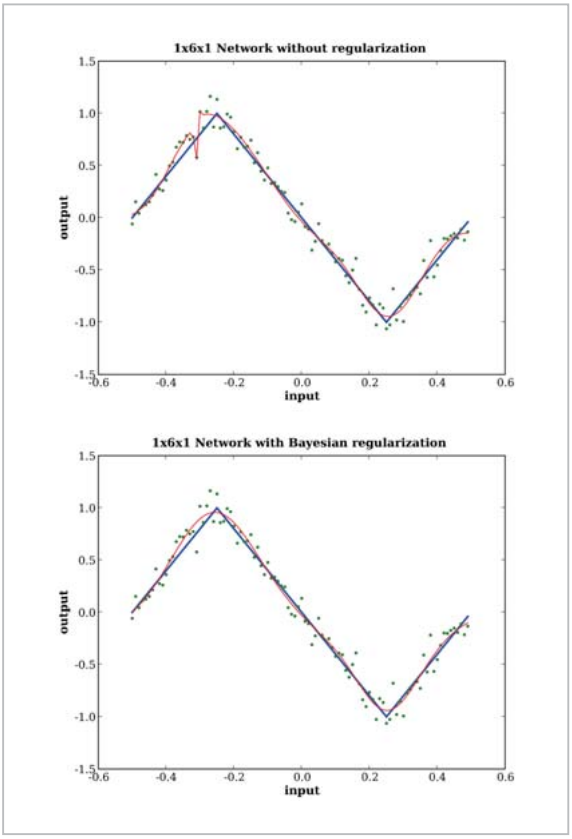

Fig.6 *Top panel: Approximation (red line) of the noisy triangle wave function (dot line) by a 1×6×1 without regularization*

The original no noisy function is also shown (blue thick line). The calculation is a remake of the one presented in [23]. Bottom panel: same as top panel but a Bayesian regularization scheme is applied.

5.3 Bayesian regularization verification

The approximation of the triangular waveform function presented in the reference where the Bayesian regularization algorithm is proposed [23], is repeated in this section.

The triangular wave function is made of 3 linear peaces between 0 and 1 (Fig. 6). The training set is a made of 100 regularly spaced data with a noise of a variance of 0.01 applied on the outputs. We have trained several $1 \times Nh \times 1$ networks with Nh set to 2,3,4,5,6,8,10 and 14. For each network, the initial value for α and β are 10^{-6} and 1, respectively. Their values are updated after each successful iteration as described in Equation 10. The initial value of the Marquardt parameter μ is set to 1; it is multiplied by 2 or divided by 10 according to the convergence status.

The top panel of the Fig. 6 shows the approximation of the function by a $1 \times 6 \times 1$ network trained without regularization term in the cost function. The overfitting effect can be seen between −0.4 and −0.2. The bottom panel shows the same approximation but using a regularized cost function and the Bayesian algorithm. The overfitting effects has been removed and the approximated function fits the original free-noisy function.

The results for the 8 trained networks are summarized in Table 7. The performance of the networks is estimated by the sum of the squared error Ea evaluated on the noisy-free training data set. The error Ea decreases down to 0.16 when the MLP size increases. For networks with more than 4 hidden neurons the error Ea becomes slightly constant. The

Table 7 *Results of the training of 8 networks of 1×Nh×1 neurons*

The number of parameters w_p is Nw and the sum of the squared error between the MLP output and the function without noise is Ea. The parameter γ is the effective number of parameters. The results of the original study [23] are given in parenthesis.

Nh	Nw	Ea	γ
2	7	0.56 (0.50)	5.6 (5.7)
3	10	0.26 (0.19)	8.7 (8.5)
4	13	0.16 (0.11)	10.2 (9.8)
5	16	0.16 (0.11)	10.3 (9.9)
6	19	0.17 (0.11)	10.3 (9.9)
8	25	0.17 (0.11)	10.3 (9.9)
10	31	0.17 (0.11)	10.3 (9.9)
14	43	0.17 (0.11)	10.4 (9.9)

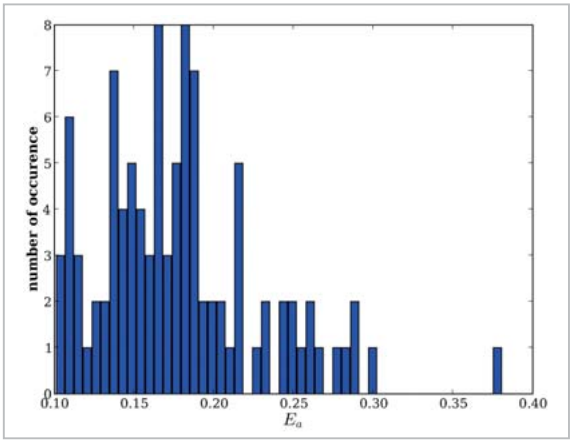

Fig.7 *Histogram of the error from a 1×6×1 network for a 100 training*

The differences between each training are the initial weights and the noise on the training data. The error on the regularized MLP output is $Ea = 0.17$.

effective number of parameter increases up to 10.3 when the number of hidden neurons increase and also becomes slightly constant for networks with more that 4 hidden neurons. The maximum value of γ corresponds to a number of hidden neurons of 3 indicating that a model with 4 hidden neurons (13 parameters) is a good choice to approximate the function. Hence, the estimated value of γ is coherent with the variation of the error Ea.

The results obtained in the original analysis, are also shown in Table 7. The value of Ea is found to be about 0.11 that is lower than the one calculated in the current analysis (0.17). Such a difference in Ea can be explained by the noise added on the training data. Figure 7 shows the histogram of the errors when a 1×6×1 MLP is trained for 100 training sets with different noise realizations. The noise has the same variance of 0.01 in each data set. Among the 100 cases, we obtain an error between 0.10 and 0.12 for about 15 cases. Hence, the results of the original analysis corresponds to the best cases of the current analysis.

6 Perspectives

A model of a feed-forward network has been implemented and described. It has been shown that the algorithms have been correctly implemented. The steepest descent algorithm used to calculate the networks parameters has to be improved. Other improvements will be included such as a better compression algorithm or the use of covariance matrices instead of scalars for the cost function regularization parameters.

However, the current state of the model allows to start the JEM/SMILES retrieval study. The next steps of the study will be to prepare a data base of simulated measurements from 10000 atmospheric scenarios and instrument noise. One part of the data base will be used in the training procedure and the other part to validate the results. A network will be set to retrieve only one parameter following the setting used for the Odin/SMR analysis. A compression level of the inputs will be defined to well represent the SMILES measurements and the best number of hidden neurons will be estimated.

The performance of the retrieval procedure will be studied in terms of vertical resolution, retrieval accuracy and sensitivity to forward model parameters errors.

Acknowledgments

P. Baron thanks the Japanese Society for Promotion of Science (JSPS) for its support.

References

1 M. Shiotani, H. Masuko, the SMILES Science Team, and the SMILES Mission Team, "JEM/SMILES mission plan. NASDA Rep. Version 2.1, NAtional Space Development Agency (NASDA), Communications Research Laboratory (CRL)", Koganei, Tokyo, 184-8795, Japan, November, 15 2002. http://smiles.tksc.jaxa.jp/indexe.shtml.

2 C. D. Rodgers, "*Inverse Methods for Atmospheric Sounding: Theory and practice*, volume 2 of *Series on Atmospheric, Oceanic and Planetery Physics*", World Scientific. Singapore-New Jersey-London-Hong-Kong, 2000.

3 Y. Kasai, C. Takahashi, S. Ochiai, P. Baron, J. Urban, T. Motoki, Y. Irimajiri, and A. Kleinboehl, "SMOCO: a retrieval code for Super-conductive Sub-Millimetr Limb Emision Sounder (SMILES) on the International Space Station", *JQSRT*, 00:00-00, 2007. submitted.

4 C. M. Bishop. "Neural networks and their applications", *Rev Sci Instrum*, 65(6):1803-1832, June 1994.

5 F. Aires, C. Prigent, W. B. Rossow, and M. Rothstein, "A new neural network approach including first guess for retrieval of atmospheric water vapor, cloud liquid water path, surface temperature, and emissivities over land from satellite microwave observations", *J. Geophys. Res.*, 106:14887-14908, 2001.

6 C. Clerbaux, J. Hadji-Lazaro, S. Payan, C. Camy-Peyret, J. Wang, D. P. Edwards, and M. Luo, "Retrieval of CO from nadir remote-sensing. measurements in the infrared by use of four. different inversion algorithms", *Applied Optics*, 41(33), 2002.

7 W. J. Blackwell, "A neural-network technique for the retrieval of atmospheric temperature and moisture profiles from high spectral resolution sounding data", *IEEE Transactions on geoscience and remote sensing*, 43:2535-2546, 2005.

8 C. Jiménez and P. Eriksson, "A neural network technique for retrieving atmospheric species from microwave limb sounders", *Radio Sci.*, 36(5):941-953, 2001.

9 C. Jiménez, "*A neural network technique for retrieving atmospheric species from microwave limb sounders*", PhD thesis, Chalmers University, Göteborg, Sweden, 2003.

10 J. Escobar Munoz, "*Base de Données pour la Restitution de Paramètres Atmosphériques à l'Échelle Globale - Étude sur l'Inversion par Réseaux de Neurones des Données des Sondeurs Verticaux Atmosphériques Satellitaires présents et à venir*", PhD thesis, Université de Paris VII, France, 1993.

11 C. Jiménez, P. Eriksson, and D. Murtagh, "First inversions of observed sub-millimetre limb sounding radiances by neural networks", *J. Geophys. Res.*, 108(24):4791, 2003.

12 M. D. Müller, A. K. Kaifel, M. Weber, S. Tellmann, J. P. Burrows, and D. Loyola, "Ozone profile retrieval from GOME data using a neural network approach (NNORSY)", *J. Geophys. Res.*, 108:4497, 2003.

13 F. Karbou, F. Aires, C. Prigent, and L. Eymard, "Potential of Advanced Microwave Sounding Unit-A (AMSU-A) and AMSU-B measurements for atmospheric temperature and humidity profiling over land", *Journal of Geophysical Research (Atmospheres)*, 110(D9):7109–+, April 2005.

14 C. Jiménez, P. Eriksson, and D. Murtagh, "Inversion of Odin limb sounding sub-millimeter observations by a neural network technique", *Radio Sci.*, 38(4):8602, 2003.

15 D. P. Murtagh, U. Frisk, F. Merino, M. Ridal, A. Jonsson, J. Stegman, G. Witt, P. Eriksson, C. Jiménez, G. Mégie, J. de La Noë, P. Ricaud, P. Baron, J. R. Pardo, A. Hauchecorne, E. J. Llewellyn, D. A. Degenstein, R. L. Gattinger, N. D. Lloyd, W. F. J. Evans, I. C. McDade, C. S. Haley, C. Sioris, C. von Savigny, B. H. Solheim, J. C. McConnell, K. Strong, E. H. Richardson, G. W. Leppelmeier, E. Kyrölä, H. Auvinen, and L. Oikarinen, "An overview of the Odin atmospheric mission", *Can. J. Phys.*, 80(4):309-318, 2002.

16 M. Van der Baan and C. Jutten, "Neural networks in geophysical applications", *Geophysics*, 65:1032-1047.

17 F. Aires, W. B. Rossow, N. Scott, and A. Chedin, “Remote sensing from the iasi instrument. 1 compression, de-noising, and first-guess retrieval algorithms”, *J. Geophys. Res.*, 107(D22), 2002.

18 F. Aires, A. Chedin, N. Scott, and W. B. Rossow, “A regularized neural network approach for retrieval of atmospheric and surface temperatures with the iasi instrument”, *Journal of Applied Meteorology*, 41(2):144-159, 2002. 14

19 M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the marquardt algorithm”, *IEEE Transactions on Neural Networks*, 5:989-993, 1994.

20 D. Nguyen and B. Widrow, “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights”, *International Joint Conference of Neural Network*, 3:21-26, 1990.

21 David J. C. MacKay, “Bayesian interpolation”, *Neural Comput.*, 4(3):415-447, 1992.

22 D. J. C. MacKay, “Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks”, *Network: Computation in Neural Systems*, 6:469-505, 1995.

23 F. D. Foresee and M. T. Hagan, “Gauss-newton approximation to bayesian learning”, In *Proceedings of the 1997 IEEE International Conference on Neural Networks.*, pages 1930-1935, 1997.

24 S. Gutjahr, “Improving the determination of the hyperparameters in bayesian learning”, In *Proceedings of the ACNN’98, Brisbane*, 1998.

25 Jan Poland, “On the robustness of update strategies for the bayesian hyperparameter alpha”, 2001. 15

***Philippe BARON**, Ph.D.*

Guest Researcher, Environment Sensing and Network Group, Applied Electromagnetic Research Center

Development of Forward and Retrieval Models for Atmospheric Remote Sensing

***Jana MENDROK**, Ph.D.*

Expert Researcher, Environment Sensing and Network Group, Applied Electromagnetic Research Center

Radiative Transfer Modeling and Cloud Remote Sensing

***KASAI Yasuko**, Dr. Sci.*

Senior Researcher, Environment Sensing and Network Group, Applied Electromagnetic Research Center

Spectroscopic Remote Sensing of Atmosphere