Data Assimilation and Retrieval Theory

Course Project: Integration of Neural Network Training into 4D Variational Data Assimilation and Application to the Lorenz Model

Andre R. Erler

December 18, 2008

Contents

1 Introduction

The atmosphere in particular but also the ocean is a complex nonlinear and chaotic system. In the case of the atmosphere reliable prediction is limited to a couple of days to a few weeks. There are mainly two factors limiting our ability to extend reliable forecasts: (a) our incomplete knowledge of the state of system (combined with its chaotic behavior), and (b) a large range of scales and insufficient understanding of the physical mechanisms governing the system (in particular at small scales). Another limitation is of course also the size of the state space, a more technical problem, which I will not consider here.

Both points listed under (b) require parameterization or approximation of unresolved or unknown processes. Standard ways of dealing with this problem are parametric (often linear) fits to observations. The first point (a) is the classic problem of data assimilation: given a set of (noisy) observations and physical/dynamical constraints, the most likely state of the system has to be estimated.

Both areas (statistical analysis and data assimilation) are closely linked and have seen several improvements in recent years. In this study I will follow the work of *Tang and Hsieh* (2001) and attempt to integrate two of the latest approaches in each field into a hybrid dynamical system.

The Lorenz model is the system I will study here. The Lorenz model has already been studied extensively and is well known for its chaotic and nonlinear behavior (Fig. 1, right, displays a typical trajectory); hence it is a common choice as a test bed for data assimilation systems and statistical analysis alike. The two major problems encountered in estimating and forecasting the state of the atmosphere can also be simulated in the Lorenz model: Uncertain and sparse observations can be simulated by adding random noise to the true trajectory before performing the analysis. The approximation of unresolved or unknown physics in a highly nonlinear system can be studied by replacing one of the dynamical equations with a suitable approximation.

In this study I replaced one component of the Lorenz model with a neural network and trained it to simulate the missing dynamical equation. This situation is meant to be analogous to e.g. the parameterization of sub–scale processes. The data assimilation scheme I will use is the 4–dimensional variational data assimilation method.

The main focus of this work will be to show how data assimilation and neural networks can be integrated in a natural way, using variational optimization methods. The hope is that both the data assimilation process and the reliability and accuracy of the neural network approximation will benefit from a successful integration.

2 Approximation of Dynamical Systems with Neural Networks

In this section I will give a brief introduction neural networks with emphasize on the use of neural networks as a method of nonlinear non–parametric function fitting. I will further describe how I used a neural network in order to approximate the behavior of one of the dynamical equations of the Lorenz model.

2.1 Brief Introduction to Neural Networks

The mathematical description of a neural network is straight forward and will be given shortly. However, what a neural network is and how it can be applied largely depends on the point of view.

Neural Networks were first studied as a model for learning processes within the (human) brain. The architecture was model to represent individual neuron and their interaction with each other;

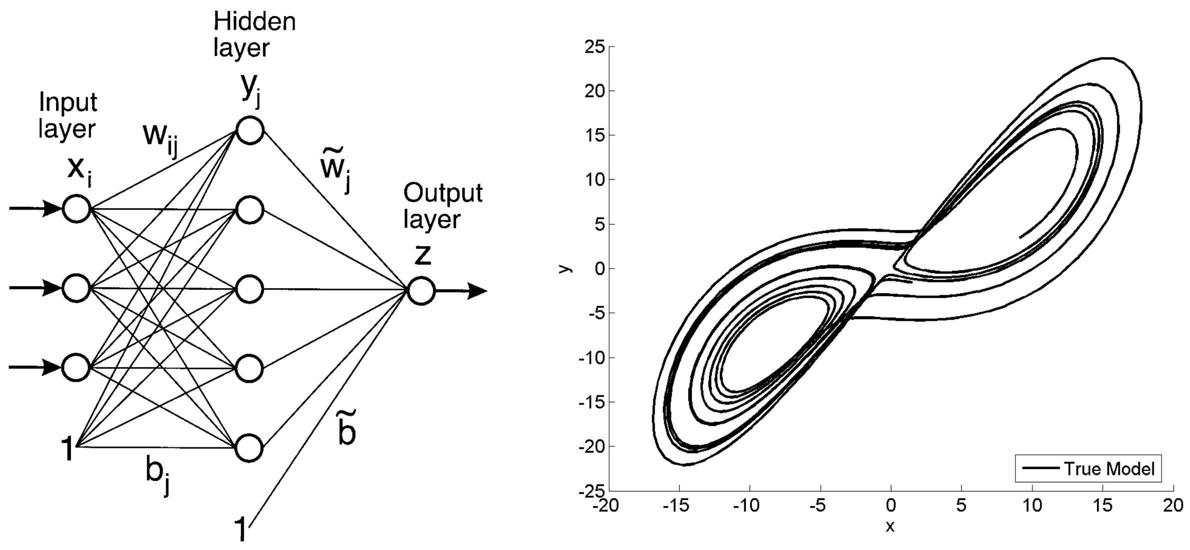

Figure 1: Left: A schematic of a simple feed forward neural network as it is used in this study. It has three input nodes, a hidden layer of five nodes and one output node. Right: A typical trajectory of the Lorenz model, covering both lobes.

hence the name.

A neural network consists of nodes (neurons) which are organized in several layers. The simplest form of a neural network is the feed forward network (cf. Fig 1, left), where every node of one particular layer can communicate with every node in the adjacent two layers. Each node has an activation level; the communication between nodes is limited to passing along the activation state to the next layer and receiving the activation state of the previous layer. The activation state of one node is determined from an (usually nonlinear) activation function, which takes the weighted sum of the activation states of all nodes of the previous layer as input. In addition each layer usually has a node with a constant activation state, which is called bias. The activation function itself is usually of sigmoid form with a near linear response in a small region and almost no response elsewhere; the two extremes, namely the linear function and the step–function are also used. The weights (connections) between nodes of adjacent layers are what defines the output of the neural network. The process of adjusting these weights in order to produce a particular desired output or behavior is called training.

Two layers of the network have a special function. The input layer is the first layer of the network and its activation state is simply the external input (which can usually take every value). The second layer with a special function is the output layer; its activation function strongly depends on the application. All other layers only communicate within the network itself; they are called hidden layers.

Beyond this simple concept there are many more sophisticated network architecture, for example networks with recurrence loops, where previous input may be fed back into the neural network at different layers. But for a large range of tasks simple feed forward networks already suffice. More complex network designs are usually used to simulate higher brain functions which are more relevant for machine learning / artificial intelligence (and modeling brain functions). Neural networks are probably most famous for their pattern recognition abilities, which are often used in image processing. Other application include associative memory and the learning of (binary and fuzzy) logic functions (with step–like activation functions for binary output).

For scientific and engineering applications however, the most common application of neural networks is function fitting/approximation: simple single layer feed forward networks can be used efficiently as non–parametric approximations/fits to arbitrary nonlinear function (naturally the number of required nodes increases with the nonlinearity of the problem).

Many applications of neural networks in oceanography and meteorology also fall into this category. In this study I will use a neural network to approximate components of the highly nonlinear Lorenz model (cf. *Tang and Hsieh*, 2001). Therefor I will limit the mathematical description to single–layer feed forward networks used for function approximation.

There is however one other major application in these fields, which is also worth mentioning: in a variation of pattern recognition, multi–layer neural networks can be used to extract (statistical) features from large data sets, in very much the same way as EOF–analysis is commonly used. The advantage of neural networks over EOF–analysis is that nonlinear relationships can be extracted (cf. *Monahan*, 2000).

Hsieh and Tang (1998) give an overview of how neural networks can be applied to meteorological and oceanographic problems. They point out three major obstacles to the application of neural networks in these fields: (a) nonlinear instability, (b) the size of data sets, and (c) the interpretation of neural network output. In the this study the size of the data set is not problematic. Interpreting the working and the output of the neural network is more relevant for pattern recognition and memory tasks and not so much a concern for function approximation. The first obstacle however will also be of major relevance to the present work: the Lorenz system is highly nonlinear and, although ensemble averaging and integration with variational data assimilation will reduce the instability somewhat, nonlinear instability and the associated poor convergence of optimization algorithms will still turn out to be one of the major limitations.

2.1.1 Mathematical Description

In Fig. 1 (left) a schematic of a simple single–layer feed forward network is displayed. The network has three input nodes, five nodes in the hidden layer and one node in the output layer. The hidden layer and the input layer each have a constant bias node. Let us denote the activation of the input layer by the vector $\mathbf{x}$, the activation of the hidden layer nodes by $\mathbf{y}$ and the output layer by $\mathbf{z}$ (which has only one component in this case). The activation of the j^{th} node of the hidden layer in terms of the input layer nodes x_i is given by

$$y_j \;=\; \sum_i \tanh\left(w_{ij}x_i \;+\; b_j\right) \;. \tag{1}$$

The activation function assumed here is the hyperbolic tangent; b_j is the connection to the constant bias node and w_{ij} describes the connection weight between the i^{th} input node and the j^{th} hidden node. The connection weights w_{ij} can be arranged into a matrix $\mathbf{W}$; the bias weights can either form a separate vector $\mathbf{b}$, or, if an additional input node with constant activation is defined, can be absorbed into $\mathbf{W}$ as an additional column. If the biases form a separate vector $\mathbf{b}$, the weights w_{ij} form a $m \times n$ matrix; m is the number of input nodes x_i and n is the number of nodes y_j in the hidden layer.

The output of the neural network is the activation of the output layer node (in general a vector);

the activation of the output node in terms of the activation levels of the hidden nodes is given by

$$z = \sum_j \tilde{w}_j y_j + \tilde{b} \tag{2}$$

Note that the activation function is linear in this case; for function approximation tasks it is common to choose a linear activation function in the output layer: otherwise the output values would be limited to a bounded interval. The nonlinearity of the network enters only in the hidden layer.

The response of the network as a whole is given by

$$z = \sum_{ij} \tilde{w}_j \tanh\left(w_{ij} x_i + b_j\right) + \tilde{b} \tag{3}$$

This simple single–layer example can of course easily be generalized to an arbitrary number of layers with different activation functions. Note however that a hidden layer with a linear activation function can immediately be absorbed into the next layer (by associativity of matrix multiplication).

The architecture described above, i.e. a single hidden layer with a nonlinear activation function and a linear output layer, is capable of approximating any nonlinear function to arbitrary precision and with a finite number of discontinuities, given a sufficient number of nodes in the hidden layer (*Cybenko*, 1989). The number of nodes in the input and output layer is determined by the dimensionality of the mapping.

The network used by *Tang and Hsieh* (2001) and in this study is the network shown in the schematic in Fig. 1 (left).

2.1.2 Training

Neural network training is the process of adjusting the weights and biases, which completely describe the model behavior, to produce the desired output, given a certain input. The desired output is usually referred to as target, while the input is call training data.

In order to quantify the performance of a given network on a given set of training data a so called error or training function is defined. A common choice is the sum of the squared errors,

$$E(\mathbf{x}, \mathbf{t}, \mathcal{N}) = \left(\mathbf{z}(\mathbf{x}, \mathcal{N}) - \mathbf{t}\right)^{\mathcal{T}} \left(\mathbf{z}(\mathbf{x}, \mathcal{N}) - \mathbf{t}\right) . \tag{4}$$

$\mathbf{x}$ is the training vector, $\mathbf{t}$ the target vector, and $\mathcal{N} = \mathcal{N}(\mathbf{W}, \tilde{\mathbf{W}}, \mathbf{b}, \tilde{\mathbf{b}})$ represents the parameters describing the neural network.

The optimal weights and biases for a given task are (by definition) those which minimize the error or training function. In this sense neural network training is merely an optimization problem. The minimization can be greatly enhanced, when the gradient of the objective function (the error function in this case) is known. This requires knowledge of the dynamics of the system, which, in the case of neural networks, is given by (3). Thus the gradient of (4) with respect to the neural network parameters $\mathbf{W}$, $\tilde{\mathbf{W}}$, $\mathbf{b}$, $\tilde{\mathbf{b}}$ can be computed. The procedure of neural network optimization using the gradient of the error function is also know as backpropagation.

Historically it was the advent of backpropagation algorithms that made the use of simulated neural networks for various computing tasks feasible. Nowadays neural networks are trained with sophisticated optimization algorithms provided as packaged software. In this study the Levenberg–Marquardt algorithm provided in the MATLAB Neural Network Toolbox was used (*Demuth et al.*, 2008).

Before I will proceed to the application to the Lorenz model, a few remarks on neural network training are in order.

The training process is probably the most important part of neural network design, and hence should be given some consideration.[1] Several factors influence the success of the training process: the convergence of the optimization algorithm, the choice of the training function, and the selection of training data.

The training data has to be chosen such that it is representative for the task that the neural network will be used for. A biased selection of training data will introduce the same bias into the networks performance: typically neural networks perform best on the task they were trained for. If the complexity of the network in comparison with the presented data is not sufficient, the network will be optimized for the type of data which occurs most frequently in the training data.

Weighting of certain components in the training function can be used to control this issue. Theoretically the training function can be used to control the training process to a very high degree, however, a too complex training function will have a detrimental impact on the convergence of the optimization algorithm; the extremely fast Levenberg–Marquardt algorithm for instance, only works with a quadratic training function of the form (4).

The convergence of the training algorithm depends on the complexity of both the training data and the training function. If the error surface is too complex the optimization likely to be trapped in a local minimum. *Hsieh and Tang* (1998) suggest to use an ensemble of identical neural networks with randomly initialized parameters, so that the training process will yield different results for each; the averaged output of the ensemble will be more robust.

Furthermore, if training data is noisy and maybe not representative for the whole input/output space, it may not even be desirable that the training algorithm converges: if the network complexity is sufficiently high, the network will fit random noise in the training data and may not perform well on slightly different data.

Having discussed the limitations of the training process, I have to emphasize, that the possibility of training itself and the availability of algorithms is actually one of the most powerful aspects of neural networks. Neural networks provide a way to approximate or infer a nonlinear relationships in an iterative manner without knowledge of the dynamics of the system.

2.2 Application to the Lorenz Model

The Lorenz model is a highly nonlinear dynamical system. It is governed by the following equations:

$$\begin{aligned} \frac{\partial X}{\partial t} &= -\sigma X + \sigma Y \\ \frac{\partial Y}{\partial t} &= -XZ + \rho X - Y \\ \frac{\partial Z}{\partial t} &= XY - \beta Z \, . \end{aligned} \tag{5}$$

σ, ρ, and β are parameters; X, Y, and Z are coordinates in phase space. The physical interpretation of the equations and parameters is not important here. The parameter values $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$ have been used. A typical trajectory of the Lorenz model covering, both lobes, is shown in Fig. 1 (right). The model was initialized at $(X, Y, Z) = (1.508870, -1.531271, 25.46091)$ and integrated for $T = 12$ (dimensionless time units).

[1] This is particularly true for more complex neural networks, such as the human brain...

In order to test the performance of neural networks as approximators for nonlinear relationships I replace the Z–component of the system with a neural network and trained the network to approximate the missing dynamical equation. The resulting hybrid model can be written as

$$\begin{aligned} \frac{\partial X}{\partial t} &= -\sigma X + \sigma Y \\ \frac{\partial Y}{\partial t} &= -XZ + \rho X - Y \\ \frac{\partial Z}{\partial t} &= \mathcal{N}(X, Y, Z) \; . \end{aligned} \tag{6}$$

This system was also studied by *Tang and Hsieh* (2001).

In the numerical implementation the Lorenz system was integrated using a 4^{th}–order Runge–Kutta scheme. The neural network was implemented in the dynamical equations, independent from the numerical scheme used.

In the next two section I will outline the training procedure and present some results.

2.2.1 Network Design and Training

The neural network used in the hybrid model (6) is the one discussed in the previous section; it is identical to the network used by *Tang and Hsieh* (2001).

The training data was generated from an ensemble of trajectories obtained from slight perturbations[2] around the initial condition $(X, Y, Z) = (-9.42, -9.43, 28.3)$ (the weakly nonlinear case in *Tang and Hsieh*, 2001). I used 25 trajectories with an integration time of $T = 15$ each and a time step of $\Delta t = 0.01$. The trajectories obtained generally stayed in one lobe of the butterfly shape but deviated somewhat.

Each network was trained with a set of 12000 consecutive data points selected from the total set of $25 \times 1500 = 37500$ data points. A total number of 25 neural networks was trained, although, unless noted otherwise, only one was actually used in the hybrid model.

In addition, to control the possibility of over–fitting one data set was generated separately and used to verify the performance of the neural networks on unknown data during the training process. Should further optimization during training turn out to have a detrimental impact on the performance on the validation data set, the training process would have been interrupted. This was never the case. The MATLAB Neural Network Toolbox also provides a Bayesian regularization algorithm (cf. *Demuth et al.*, 2008), which provides an estimate of the degrees of freedom actually used by the network: the result was that the networks used all their available degrees of freedom. I conclude, given the selection of training data, over–fitting was not a major problem. For a larger variety of training data it would in fact be appropriate to increase the network complexity.

The range of the training data chosen here may appear rather limited compared with the whole phase space, and of course a greater variety in the training data will lead to a better generalization. I purposely restricted the system to the weakly nonlinear regime in order to avoid the poor convergence of training and data assimilation algorithms (section 3) in the strongly nonlinear regime (cf. *Tang and Hsieh*, 2001).

Pre– and Post Processing The data was preprocessed by removing the mean and normalizing to one standard deviation (a common procedure). I also tested a smaller neural network architecture

[2]The perturbations where evenly distributed in the interval $[0, 0.1]$.

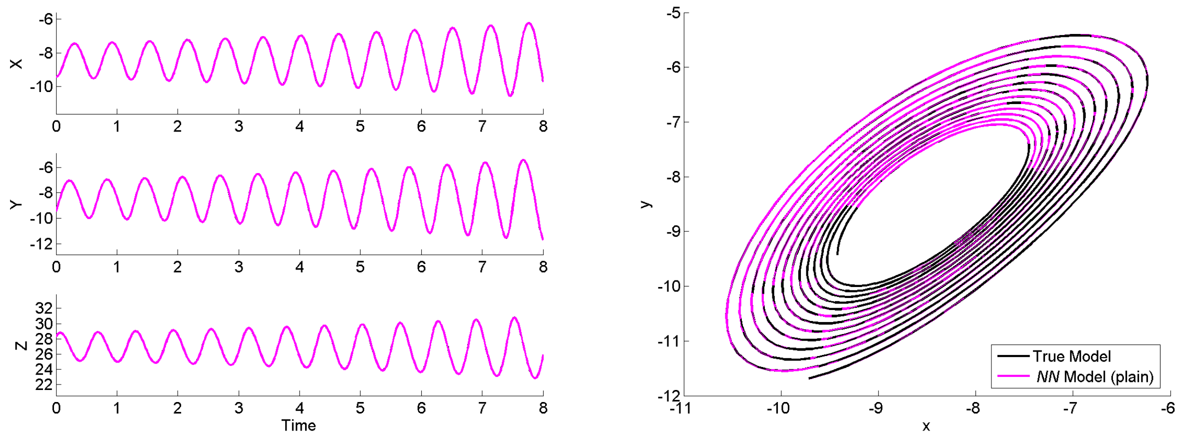

Figure 2: A typical training trajectory in the neighborhood of the training data, integrated for $T = 8$. The left panel displays each component separately, the right panel shows a projection onto the $X - Y$–plane. The purple line is the hybrid model, the black line the true Lorenz model (the black line is entirely covered by the purple line on the left panel).

in combination with principal component analysis and truncation of the smallest component (<2% total variance); thus the dimension of the input space was reduced to the two leading principal components (instead of three), which reduces the parameter space of the neural network by five. I can not report any significant change in network performance.

The reason is that the PCA–analysis (with truncation) can also be expressed as a 3×2 projection matrix P and connects linearly to the input layer. The matrix product of the PCA–matrix P and the reduced weight matrix $\mathbf{W}^{PCA}$ will have the same dimension as the weight matrix described above,

$$\mathbf{W} \approx \mathbf{W}^{PCA} \times \mathbf{P} . \tag{7}$$

It is then fair to assume that the optimization algorithm will find the correct weights to simulated the PCA–analysis, provided the PCA–analysis makes sense in the first place (it certainly does: the fractal dimension of the Lorenz system is only about 2.04, cf. *Monahan*, 2000).

2.2.2 Preliminary Results

Now it is time to see how the networks perform. A typical training trajectory in the neighborhood of the training data is displayed in Fig. 2. The trajectory was integrated over $T = 8$. Evidently the neural network approximation is near perfect.

Fig. 3 shows two integrations from identical initial conditions, $(X, Y, Z) = (-14, -6.1, 25.7)$, for two different hybrid models: The lower row is a standard model with one neural network replacing the Z–component; the upper row shows the trajectory of a hybrid model using an ensemble of 25 neural networks approximating the Z–component. Evidently the generalization and stability of the ensemble is significantly better that that of a single network. The behavior displayed in the bottom row is typical for a number of neural networks and many different initial conditions; although not all networks exhibit the instability with the initial conditions given above, it is very like that most networks will fail under some conditions. The reason is the nonlinearity of the system: The neural network training algorithm frequently ends up in a local minimum. With an ensemble of

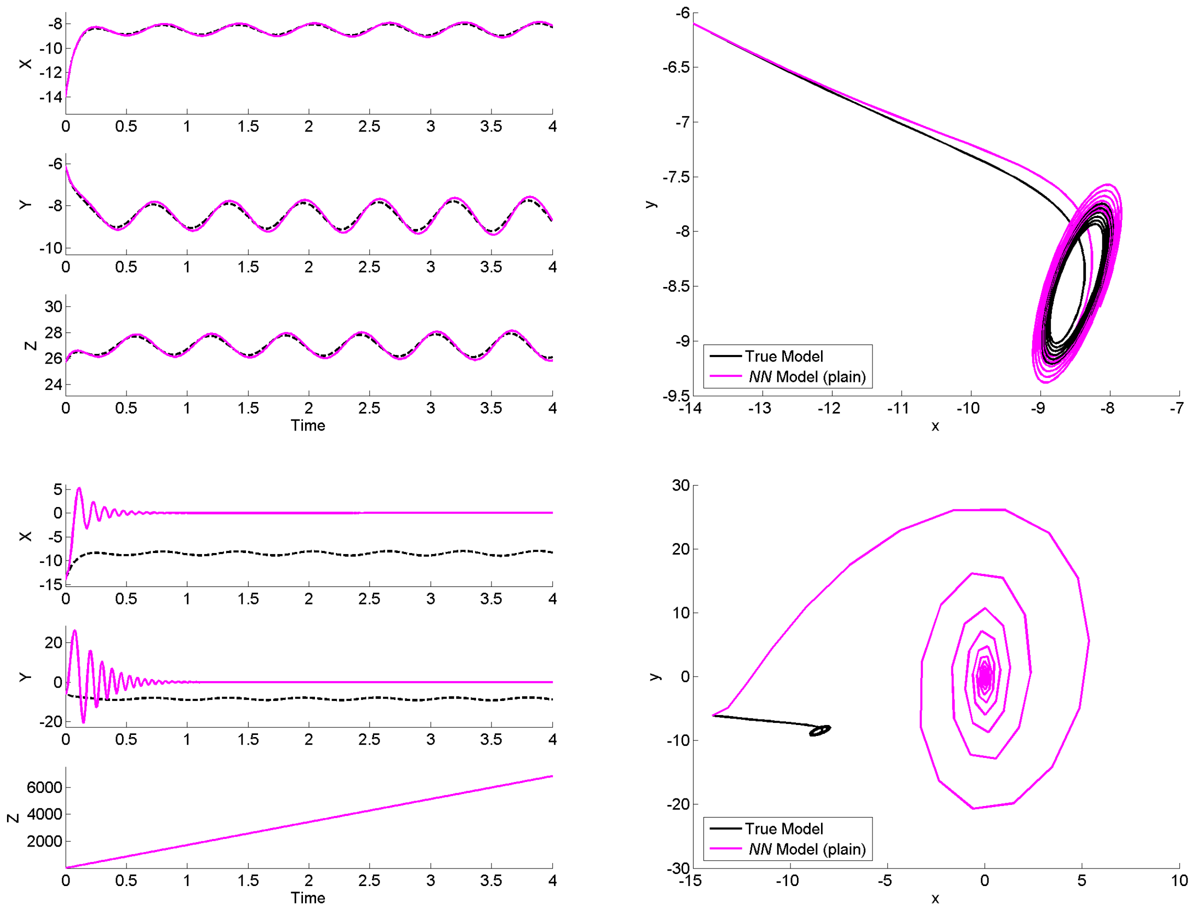

Figure 3: The upper row shows the trajectory of a hybrid model using an ensemble of 25 neural networks, the lower row shows the same integration (same initial conditions) for a hybrid model using only one neural network. The organization of the plots is otherwise identical to Fig. 2

neural networks the local minima will mostly be scattered around the global minimum and the generalization and stability will be much better (*Hsieh and Tang*, 1998).

Also note that it is the Z–component of the hybrid model with only one neural network, which deviates the most, while the others appear to remain fairly stable. This does not agree with the observations of *Tang and Hsieh* (2001), but it is possible that my results are not (yet) statistically significant. This issue needs more investigation.

3 Application of 4D-Var to the Lorenz Model

The second step towards the integration of neural networks into variational data assimilation is of course to implement a variational data assimilation algorithm; in this section I will review relevant aspects of the theory of variational data assimilation and describe the numerical implementation and application to the Lorenz model. The data assimilation algorithm used in this study was 4D-Var.

3.1 Variational Data Assimilation

The problem addressed in data assimilation is the estimation of the state of a system given a set of observations (and a background / a priori estimate). The best estimate of the state of the system is the state with the smallest deviation from the observations (and the background).

The approach of variational data assimilation is to measure the deviation of the state estimate from the observations (and the background) by defining a cost function $J(\mathbf{v}, \mathbf{z})$ (a function of the estimated state $\mathbf{v}$, and the observations $\mathbf{z}$; for the purpose of this discussion the background will be treated as part of the observations, and will be omitted henceforth). A common choice for the cost function is the sum of the mean square errors:

$$J(\mathbf{v}, \mathbf{z}) \;=\; (\mathbf{v} - \mathbf{z})^T \, \mathbf{W} \, (\mathbf{v} - \mathbf{z}) \;, \tag{8}$$

where $\mathbf{W}$ is a weight matrix, often chosen to be the inverse of the observation error covariance matrix $\mathbf{W} = \mathbf{R}^{-1}$. By defining the cost function J the task of data assimilation reduces to an optimization problem where the cost function is to be minimized under a given set of constraints. The (known) dynamics of the system enter in the specification of the constraints: in the case of 3D data assimilation (at one time step) these may be diagnostic balance conditions; in the 4–dimensional case (including time), however, the dynamical evolution of the system over time can be taken into account.

In classical mechanics the dynamical evolution of a system is formulated as a minimization problem, where the temporal integral of the Action is to be minimized. More generally, Lagrange multipliers can be introduced to define the Action of a constrained dynamical system, and thus find the constrained trajectory.

Consider the following general form of a dynamical system:

$$\frac{\partial \mathbf{v}}{\partial t} \;=\; \mathbf{f}(\mathbf{v}, \mathbf{p}, t) \tag{9}$$

$\mathbf{v}$ is the state vector, $\mathbf{p}$ is a parameter vector[3], and t the explicit time dependence. For the (unconstrained) dynamical system (9) the Lagrangian can be written as

$$\mathbf{L}(\mathbf{v}, \dot{\mathbf{v}}, \lambda) \;=\; \int_{t_0}^{T} \lambda \left(\frac{\partial \mathbf{v}}{\partial t} \;-\; \mathbf{f}(\mathbf{v}, \mathbf{p}, t) \right) \, dt \;. \tag{10}$$

$\dot{\mathbf{v}} = \partial \mathbf{v} / \partial t$ is the partial derivative with respect to time, and λ is the vector of Lagrange multipliers. Classically the minimum of (10) is exactly zero, $L = 0$. Variational calculus can be used to derive the dynamical equations which minimize L. In the trivial case above this will simply yield the dynamical equations (9).

The power of the formulation with Lagrange multipliers is that additional constraints can be introduced, and the minimum of the generalized Lagrangian will yield the trajectory of the dynamical system under the given constraints. Again, methods from variational calculus can be employed to retrieve the dynamical equations (*Polavarapu*, 2008).

It is now straight forward to treat the cost function J as such a constraint and minimize the generalize Lagrangian given by

$$\mathbf{L}(\mathbf{v}, \dot{\mathbf{v}}, \lambda) \;=\; J(\mathbf{v}, \mathbf{z}) \;+\; \int_{t_0}^{T} \lambda^{\dagger} \left(\frac{\partial \mathbf{v}}{\partial t} \;-\; \mathbf{f}(\mathbf{v}, \mathbf{p}, t) \right) \, dt \;. \tag{11}$$

[3]Typically parameters are fixed, but for the purpose of data assimilation I will relax this condition in section 4.1.

At this point it facilitates the physical interpretation when the Lagrange multipliers λ are replaced by $\mathbf{v}^\dagger$, which will henceforth be referred to as the adjoint model (corresponding to the dynamical model $\mathbf{v}$, which is actually the trajectory).

In variational data assimilation we are interested in minimizing the cost function under the given dynamical constraint (9). Minimizing (11) will do exactly that. At a (local) minimum the (first order) variation of the Lagrangian $\delta\mathbf{L}$ has to vanish. A number of useful relationships, which facilitate the optimization process, can be derived by considering partial derivatives of the Lagrangian: partial derivatives with respect to orthogonal coordinates and parameters have to vanish individually to satisfy the minimization condition of the Lagrangian.

Integrating (11) by parts yields

$$\mathbf{L}(\mathbf{v},\dot{\mathbf{v}},\mathbf{v}^\dagger) \;=\; J(\mathbf{v},\mathbf{z}) \;+\; \Big[\mathbf{v}^\dagger\mathbf{v}\Big]_{t_0}^{T} \;-\; \int_{t_0}^{T}\left(\frac{\partial\mathbf{v}^\dagger}{\partial t}\mathbf{v} \;+\; \mathbf{v}^\dagger\mathbf{f}(\mathbf{v},\mathbf{p},t)\right)\,dt\;. \tag{12}$$

Consider the variation of (12) with respect to the initial state $\mathbf{v}_0 = \mathbf{v}(t_0)$

$$\delta\mathbf{L} \;=\; \nabla_{\mathbf{v}_0}J(\mathbf{v},\mathbf{z})\delta\mathbf{v}_0 \;+\; \mathbf{v}_0^\dagger\delta\mathbf{v}_0\;. \tag{13}$$

Requiring $\delta\mathbf{L}/\delta\mathbf{v}$ to vanish, yields

$$-\nabla_{\mathbf{v}_0}J(\mathbf{v}_0,\mathbf{z}) \;=\; \mathbf{v}^\dagger(t_0)\;. \tag{14}$$

The (negative) gradient of the cost function J with respect to the initial conditions $\mathbf{v}_0$ under the given dynamical constraints is given by the state of the adjoint model $\mathbf{v}^\dagger$ at $t = t_0$.

In order to obtain an useful expression for the adjoint model, we have to consider the form of the cost function in the 4–dimensional case: it is the integral of (8) over the assimilation period,

$$J(\mathbf{v},\mathbf{z}) \;=\; \int_{t_0}^{T}(\mathbf{v}-\mathbf{z})^T\,\mathbf{W}\,(\mathbf{v}-\mathbf{z})\;dt \;=\; \int_{t_0}^{T}\mathbf{D}(\mathbf{v},\mathbf{z})\,dt\;, \tag{15}$$

(where $D(\mathbf{v},\mathbf{z})$ was introduced as a shorthand notation, cf. *Tang and Hsieh*, 2001)

Substituting (15) into (12) yields

$$\mathbf{L}(\mathbf{v},\dot{\mathbf{v}},\mathbf{v}^\dagger) \;=\; \Big[\mathbf{v}^\dagger\mathbf{v}\Big]_{t_0}^{T} \;+\; \int_{t_0}^{T}\left(\mathbf{D}(\mathbf{v},\mathbf{z}) \;-\; \frac{\partial\mathbf{v}^\dagger}{\partial t}\mathbf{v} \;-\; \mathbf{v}^\dagger\mathbf{f}(\mathbf{v},\mathbf{p},t)\right)\,dt\;. \tag{16}$$

The first order variation of (16) with respect to $\mathbf{v}^\dagger$ will simply retrieve the dynamical system (9). The variation with respect to $\mathbf{v}$, however, yields

$$\delta\mathbf{L} \;=\; \Big[\mathbf{v}^\dagger\delta\mathbf{v}\Big]_{t}^{T} t_0 \;+\; \int_{t_0}^{T}\left(\nabla_{\mathbf{v}}\mathbf{D}(\mathbf{v},\mathbf{z})\delta\mathbf{v} \;-\; \frac{\partial\mathbf{v}^\dagger}{\partial t}\delta\mathbf{v} \;-\; \mathbf{v}^\dagger\nabla_{\mathbf{v}}\mathbf{f}(\mathbf{v},\mathbf{p},t)\delta\mathbf{v}\right)\,dt\;. \tag{17}$$

Requiring $\delta\mathbf{L}/\delta\mathbf{v}$ to vanish, we can immediately read off the dynamical equations for the adjoint model (the Lagrange multipliers), including the initial conditions:

$$-\frac{\partial\mathbf{v}^\dagger}{\partial t} \;=\; \nabla_{\mathbf{v}}\mathbf{D}(\mathbf{v},\mathbf{z}) \;-\; \mathbf{v}^\dagger\nabla_{\mathbf{v}}\mathbf{f}(\mathbf{v},\mathbf{p},t)\;, \tag{18}$$

$$\mathbf{v}^\dagger(T) \;=\; 0\;. \tag{19}$$

Essentially (18) is a dynamical system with an additional forcing term $\nabla_{\mathbf{V}}\mathbf{D}(\mathbf{v},\mathbf{z})$. The initial condition (19) is taken at $t = T$ because the optimization is usually performed with respect to the initial state of the system $\mathbf{v}_0$ at $t = t_0$ (cf. Eq. 14).

The gradient obtained in (14) can now be used to minimize the cost function $J(\mathbf{v}_0, \mathbf{z})$ as a function of the initial conditions; but before we can evaluate the gradient (14), we have to integrate the adjoint model from $t = T$ backwards in order to obtain the Lagrange multiplier (adjoint model state) at $t = t_0$.

In fact exactly this is done in discretized form in the standard implementation of the 4D-Var algorithm, as described by *Bouttier and Courtier* (1999).

3.2 Implementation of 4D-Var

I implemented the 4D-Var algorithm following the description in *Bouttier and Courtier* (1999).

The code for the Lorenz model itself, as well as for the tangent linear model and its adjoint was provided. Before I implemented the algorithm, however, I performed a test of the adjoint. The test contained in the provided code did in fact not test the adjoint linear model. I tested the adjoint by integrating the tangent linear model forward along a given trajectory and backward again. The resulting state was identical to the initial state up to machines precision.

The integration scheme for the full Lorenz model was the 4^{th}–order Runge–Kutta scheme (cf. *Tang and Hsieh*, 2001). Accordingly the tangent linear and the adjoint were derived from the numerical implementation of the Lorenz system.

In order to implement 4D-Var, I had to define a cost function, as well as a gradient of the cost function. The cost function used here is

$$J(\mathbf{v}(t_i), \mathbf{z}(t_i)) \;=\; \sum_i (\mathbf{v}_i - \mathbf{z}_i)^T \, \mathbf{W} \, (\mathbf{v}_i - \mathbf{z}_i) \ , \tag{20}$$

where the sum is to be taken over all time steps of the assimilation window.

The gradient of the cost function with respect to the initial conditions is given in (14). In discrete form it can be written as

$$-\frac{1}{2} \nabla_{\mathbf{v}_0} J \;=\; \sum_i \left[\prod_j^i \left(\mathbf{M}_j^\dagger \right) \mathbf{R}^{-1} (\mathbf{v}_i - \mathbf{z}_i) \right] \ , \tag{21}$$

where $\mathbf{M}^\dagger$ is the discrete form of the adjoint model (*Bouttier and Courtier*, 1999). Note that the forward model, which transforms from state space to observation space, has been omitted here because all variables are “observed” directly.

With the gradient obtained from (21), I used a quasi–Newton scheme with several iterations to find the optimal initial conditions $\mathbf{v}_0$. However, due to the nonlinearity of the problem the algorithm did not always converge. Convergence was particularly poor in the highly nonlinear regime.

3.3 Some Results

Figs. 4 & 5 shows results from the 4D-Var data assimilation algorithm described above. The black line is the true trajectory, and the blue crosses indicate the points where observations were available (every time step). The observations were obtained from the true trajectory by adding Gaussian noise. Only the observations were available to the 4D-Var algorithm to produce the analysis. The

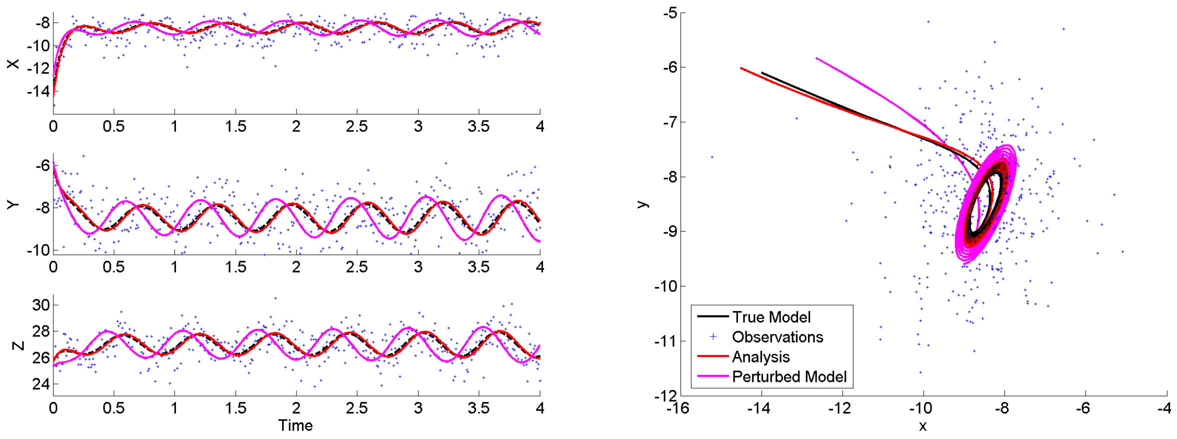

Figure 4: Results of the 4D-Var assimilation in the weakly nonlinear regime. The black line is the true model state and the blue crosses are observations (observation error $\sigma = 1$); the red line the analysis produced by the 4D-Var algorithm, and the purple line is the trivial forecast, integrated from the first observation.

analysis is indicated by the red line; the purple line is a model integration starting from the first observation point (to illustrate the skill of the 4D-Var algorithm as compared to the trivial solution). The integrations shown in Fig. 4 is identical in setup to the integrations shown in Fig. 3; note however that the model used in Fig. 4 is the full Lorenz model, not the hybrid model. The standard deviation of the observations is $\sigma = 1$, the observation window was 200 time steps, and the integration time was $T = 4$.

Fig. 5 shows a data assimilation run in the highly nonlinear region; the initial conditions were $(X, Y, Z) = (1.508870, -1.531271, 25.46091)$; the observation error was again $\sigma = 1$ and the assimilation window was 400 time steps, the major difference, is that a smaller time step of $\Delta t = 0.001$ over an integration time of only $T = 2$ was used.

Experiments have shown that the analysis generally fares better when the assimilation window is short (in time) with an optimal number of data points of roughly 400. These results are consistent with the findings of *Tang and Hsieh* (2001) (although they only reported results for the integrated hybrid model).

I imposed a weak continuity constraint on the solutions from adjacent assimilation windows by overlapping the assimilation windows and passing on the last data point of the previous assimilation as the initial observation for the next assimilation window.[4]

In all cases it was evident, that the data assimilation algorithm shows superior skill and stability over a longer periods of time than the trivial solution, although the algorithm does not always converge to the optimal solution.

[4]In theory the continuity constraint could be varied by adjusting the error covariance for the first observation, but this was not done here; the results of such a procedure were generally not very good.

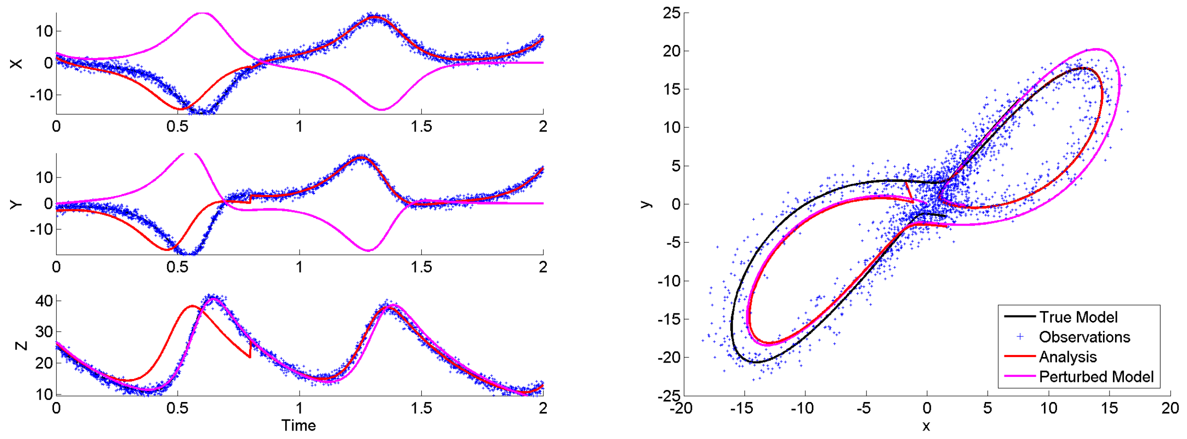

Figure 5: Results of the 4D-Var data assimilation scheme in the highly nonlinear regime; plot organization and annotation as well as model setup is similar to Fig. 4, except that different initial conditions, a smaller time step, and a larger assimilation window were used.

4 Integration of Neural Network Training into 4D Variational Data Assimilation

The final step in the work presented here was to integrate the training process of neural networks into the 4D-Var algorithm. Both, neural network training and variational data assimilation are essentially optimization problems where the objective is, to find a trajectory that fits the observations (targets) as close as possible. The analogy of neural network training and 3D-Var is immediately obvious. Furthermore, considering that 4D-Var is only an extension of 3D-Var which accounts for the dynamical constraints of the system, it is reasonable to expect neural network training to benefit from a similar approach (which respects dynamical constraints).

Tang and Hsieh (2001) reported that in experiments where conventionally trained neural networks were used in the hybrid model, the Z–coordinate was typically reasonably well approximated, while small errors introduced by the neural network approximation caused the X– and Y–component to diverge quickly. The experiments conducted by myself and discussed in section 2.2, however, do not support their findings. Nevertheless it will be a worthwhile exercise to see whether a dynamically constraint training algorithm will improve the neural network performance in highly nonlinear systems.

In order to integrate neural network training with variational data assimilation, two steps are important: (a) the structure of the neural network (weights and biases) has to be mapped to parameters of the (hybrid) dynamical model, and (b) the optimization of those parameters has to be included in the data assimilation scheme. Also note that the cost function (8) used in variational data assimilation and the error function (4) used in neural network training have the same mathematical form when the observations are interpreted as targets for the backpropagation training algorithm. It is hence not even necessary to reformulate the Lagrangian (11).

I will start with a general formulation of the parameter estimation problem in variational data assimilation, before I will discuss how neural network parameters in particular can be optimized. The main task will be to derive an analytic expression for the gradient of the cost function in terms of the parameters to be optimized.

4.1 Parameter Estimation using 4D-Var

Variational data assimilation can not only be used to estimate optimal initial conditions but also to estimate optimal values for model parameters (or both simultaneously). To this end model parameters can simply be treated as free variables of the Lagrangian (11); for the training algorithm to be effective, however, it is also necessary to derive an analytical expression for the gradient of $J(\mathbf{v}, \mathbf{p}, \mathbf{z})$ with respect to the parameters to be optimized. An approach similar to the one taken in section 3.1 will also be successful here: consider the first order variation of the Lagrangian (11) with respect to the parameter vector $\mathbf{p}$

$$\delta\mathbf{L}(\mathbf{v}, \mathbf{p}, \dot{\mathbf{v}}, \mathbf{v}^\dagger) \;=\; \nabla_{\mathbf{p}} J(\mathbf{v}, \mathbf{p}, \mathbf{z})\delta\mathbf{p} \;+\; \int_{t_0}^{T} \mathbf{v}^\dagger \nabla_{\mathbf{p}} \mathbf{f}(\mathbf{v}, \mathbf{p}, t)\delta\mathbf{p}\, dt \tag{22}$$

and require $\delta\mathbf{L}/\delta\mathbf{p}$ to vanish. The (negative) gradient of the cost function with respect to the model parameters is then

$$-\nabla_{\mathbf{p}} J(\mathbf{v}, \mathbf{p}, \mathbf{z}) \;=\; \int_{t_0}^{T} \mathbf{v}^\dagger \nabla_{\mathbf{p}} \mathbf{f}(\mathbf{v}, \mathbf{p}, t)\, dt \;. \tag{23}$$

The particular form of the gradient depends on the gradient of the dynamical system with respect to its parameters. In section 4.2 I will derive the gradient for the hybrid model with respect to neural network parameters.

Before (23) can be evaluated for the discretized model, however, some more work is necessary.

The numerical integration scheme used here is the 4^{th}–order Runge–Kutta scheme. For a general dynamical system it can be written as follows:

$$\begin{aligned}
\mathbf{v}_{i+1} &= \mathbf{v}_i + \frac{\mathbf{k1}_i}{6} + \frac{\mathbf{k2}_i}{3} + \frac{\mathbf{k3}_i}{3} + \frac{\mathbf{k4}_i}{6} \\
\mathbf{k1}_i &= \Delta t\, \mathbf{f}(\mathbf{v}_i, \mathbf{p}, t_i) \\
\mathbf{k2}_i &= \Delta t\, \mathbf{f}(\mathbf{v}_i + \mathbf{k1}_i, \mathbf{p}, t_i + \Delta t/2) \\
\mathbf{k3}_i &= \Delta t\, \mathbf{f}(\mathbf{v}_i + \mathbf{k2}_i, \mathbf{p}, t_i + \Delta t/2) \\
\mathbf{k4}_i &= \Delta t\, \mathbf{f}(\mathbf{v}_i + \mathbf{k3}_i, \mathbf{p}, t_i + \Delta t)\;.
\end{aligned} \tag{24}$$

The gradient of the discrete time–evolution operator $\mathbf{K}_i = \mathbf{k1}_i/6 + \mathbf{k2}_i/3 + \mathbf{k3}_i/3 + \mathbf{k4}_i/6$ can now be derived from (24) by applying the chain rule and inserting the components into (23). The gradient of $\mathbf{K}_i$ can most conveniently be computed in a recursive manner:

$$\begin{aligned}
\frac{\partial \mathbf{K}_i}{\partial \mathbf{p}} &= \frac{\partial \mathbf{k}1_i}{\partial \mathbf{p}} + \frac{\partial \mathbf{k}2_i}{\partial \mathbf{p}} + \frac{\partial \mathbf{k}3_i}{\partial \mathbf{p}} + \frac{\partial \mathbf{k}4_i}{\partial \mathbf{p}} \\
\frac{\partial \mathbf{k}1_i}{\partial \mathbf{p}} &= \frac{\partial \mathbf{f}_i}{\partial \mathbf{p}} \\
\frac{\partial \mathbf{k}2_i}{\partial \mathbf{p}} &= \frac{\partial \mathbf{f}_i}{\partial \mathbf{p}} \left(1 + \frac{1}{2}\frac{\partial \mathbf{k}1_i}{\partial \mathbf{p}}\right) \\
\frac{\partial \mathbf{k}3_i}{\partial \mathbf{p}} &= \frac{\partial \mathbf{f}_i}{\partial \mathbf{p}} \left(1 + \frac{1}{2}\frac{\partial \mathbf{k}2_i}{\partial \mathbf{p}}\right) \\
\frac{\partial \mathbf{k}4_i}{\partial \mathbf{p}} &= \frac{\partial \mathbf{f}_i}{\partial \mathbf{p}} \left(1 + \frac{\partial \mathbf{k}4_i}{\partial \mathbf{p}}\right)\;.
\end{aligned} \tag{25}$$

With the above relation ships the discrete form of the (negative) gradient of J with respect to arbitrary parameters is given by

$$-\nabla_{\mathbf{p}} J(\mathbf{v}_i, \mathbf{p}, \mathbf{z}_i) \;=\; \sum_i \mathbf{v}_i^\dagger \nabla_{\mathbf{p}} \mathbf{K}(\mathbf{v}_i, \mathbf{p}, t_i)\;, \tag{26}$$

where $\mathbf{v}_i^\dagger$ is again the discrete form of the adjoint model and $\nabla_{\mathbf{p}} \mathbf{K}_i$ is the gradient time evolution operator given in (25). For more details on the discrete formulation of the variational data assimilation see *Lu and Hsieh* (1998) or *Tang and Hsieh* (2001), appendix B.

4.1.1 Estimation of Dynamical Parameters

In order to test the parameter estimation algorithm, I conducted a number of experiments where the parameters σ, ρ, and β of the Lorenz system (5) were perturbed and had to be retrieved by the 4D-Var algorithm. The gradient $\partial\mathbf{f}/\partial\mathbf{p}$ is simply a diagonal matrix with elements $(Y - X, X, Z)$.

Under conditions with perfect observations, the parameters were reliably retrieved by the 4D-Var algorithm from an initial perturbation of up to $\sigma = 0.5$ standard deviations. When an observation error was added, both, initial conditions and parameters, could be retrieved up to a standard deviation of $\sigma = 0.2$. In both cases (with and without observation error) the retrieval was only performed for the weakly nonlinear case.

It is also interesting to note that the parameter estimate in the case with observation errors was generally more accurate when initial conditions were retrieved at the same time. This is not very surprising: In fact there is no way the assimilation algorithm can reproduce the correct trajectory from perturbed initial conditions without altering the parameters with respect to the true value (there are no other degrees of freedom); hence with imperfect observations a simultaneous retrieval will yield more realistic results.

Another observation worth reporting is that parameter retrievals performed over consecutive assimilation windows will usually converge to the exact value (up to machine precision) within only a few assimilation windows (typically after two to three 400–time step windows; the last retrieval was used as the background for the next assimilation window).

4.2 Neural Network Optimization with 4D-Var

After a general form for parameter estimation within the 4D-Var scheme was derived in section 4.1, I will now show how the neural network parameters of the hybrid model (6) can be included in the optimization process. To this end the gradient of the hybrid dynamical system with respect to the neural network weights and biases has to be derived and implemented.

4.2.1 The Gradient of the Cost Function

According to (25) the discrete gradient (26) can be expressed in terms of the analytic gradient of the dynamical system. The system under consideration is the hybrid model (6) with the Z–component given by the neural network response (3). Using the relation $\frac{\partial}{\partial x}\tanh x = 1 - \tanh^2 x$ and y_j as given by (1), the gradient of (3) with respect to the weights and biases is

$$\frac{\partial z}{\partial \tilde{w}_j} = y_j \tag{27}$$

$$\frac{\partial z}{\partial \tilde{b}} = 1 \tag{28}$$

$$\frac{\partial z}{\partial w_{ij}} = \tilde{w}_j \frac{\partial}{\partial w_{ij}} y_j = \tilde{w}_j \left(1 - y_j^2\right) x_i \tag{29}$$

$$\frac{\partial z}{\partial w_{ij}} = \tilde{w}_j \frac{\partial}{\partial b_j} y_j = \tilde{w}_j \left(1 - y_j^2\right) . \tag{30}$$

The form of the gradient given by (27) - (30) can now be implemented into (26); mind however, that the appropriate normalization factors used for pre– and post–processing have to be applied as well.

Here I want to mention that, in theory, the integration of the hybrid model into the 4–dimensional variational data assimilation scheme requires yet another adaption: the hybrid model is technically not identical to the discrete form of the exact Lorenz model. Hence the tangent linear and the adjoint model also take on a different form. However, given severe time constraints and the complexity of even a small neural network, I did not attempt to derive the correct adjoint model for the hybrid model; in absence of a suitable TAMC (Tangent and Adjoint Model Compiler) I used the adjoint model of the discretized full Lorenz model (5). The results will turn out to be quite reasonably, which justify this approximation.

4.3 Results

Fig. 6 shows the results of 4D-Var parameter estimation on a hybrid model. The initial conditions were the same as in Fig. 3 and the network used was the unstable ensemble members shown in the bottom row of Fig. 6. The red trajectory is the successful analysis with optimized weights and biases, the purple trajectory is a different ensemble member which is more stable in this region. In this case the analysis has successfully overcome the instability of the ensemble member. It is however important to note that this is not always the case: again the convergence of the optimization algorithms is not very reliable and deteriorates with increasing distance from the training data set.

Close to the training data set the assimilation is very reliable; the initial weights and biases of the neural networks are good enough, so that it is even possible to perform a real assimilation run with the hybrid model: Fig. 7 displays the analysis trajectory of a hybrid model initialized close to the training data (red line). The trajectory without assimilation (purple) orbits the attractor at a too large radius as compared to the true trajectory (black line). By "real assimilation" I mean assimilation of noisy data: the standard deviation of the observations was $\sigma = 0.2$, the assimilation window 200 time steps and the integration time $T = 4$. The hybrid model does fairly well on the noisy data and the assimilation scheme does have a better skill than the trivial estimate (purple line). It does however not do as good a job as the full Lorenz model when the observation error is increased. Presumably the model error introduced by the hybrid model reduces the tolerance with respect to observation errors.[5]

A general observation is that the greatly increased number of parameters due to the neural network training (26 in total) makes the optimization process significantly slower and less reliable;

[5]In fact the 4D-Var algorithm as implemented here assumes that the model is perfect; by construction the hybrid model is not perfect and thus the analysis retrieved here is in fact not the optimal estimate of the true state.

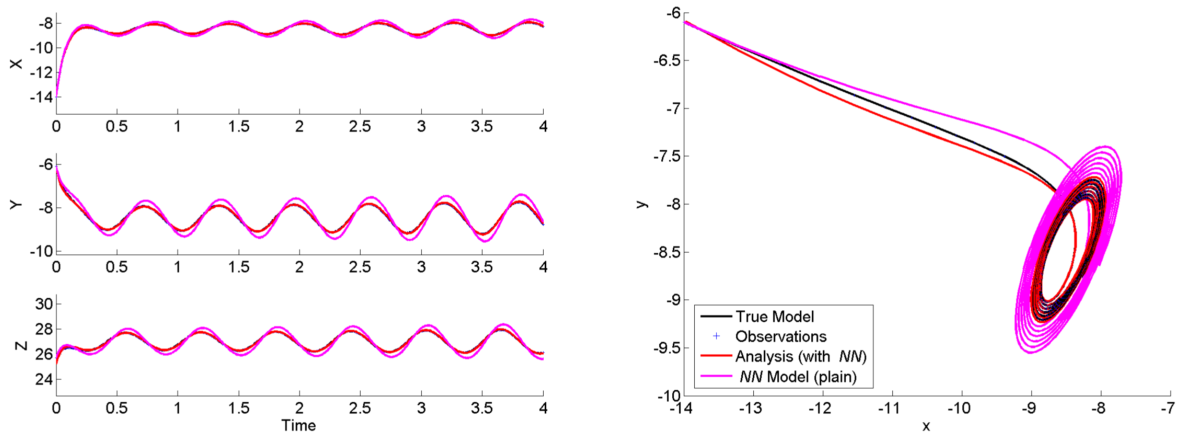

Figure 6: A neural network trained on–line with by the 4D-Var parameter estimation algorithm. The initial conditions and the network are the same as in Fig. 3 (lower row); The organization and annotation as well. The read line is the trajectory optimized by 4D-Var, the purple line is the trajectory of a different (more stable) ensemble member.

the larger the number of parameters, the poorer the convergence. Experiments with a assimilation setup where only the output layer of the neural network was optimized, in general showed better convergence. Varying the number of data points (i.e. the size of the assimilation window and the observation frequency) did not lead to better performance or convergence. Training an ensemble within the 4D-Var optimization was not feasible.

Also I want to add that the optimization performed by the 4D-Var algorithm (in the case of convergence) does usually not seem to increase the overall skill of the hybrid model. In most cases a hybrid model not previously trained under 4D-Var data assimilation (or reinitialized thereafter) performed better and was more stable than a hybrid model previously optimized under 4D-Var assimilation (on a different assimilation window).

5 Summary and Conclusion

In this study I have used neuronal networks to approximate the Z–component of the Lorenz system. The neural networks used were simple feed forward networks with 5 hidden nodes (one layer).

I used the full Lorenz model to generate data and train an ensemble of neural networks. I chose to limit the training data to an area within one of the two lobes, where the Lorenz model exhibits only weak nonlinearity. The trajectories produced by the hybrid model matched the true trajectories in the vicinity of the training data almost perfectly, but the agreement deteriorated further away from the training data. Far away from the training data the neural networks began to exhibits instabilities, where errors in the neural network approximation caused the trajectory to diverge very far from the true trajectory (a typical nonlinear/chaotic effect of the Lorenz model).

I also implemented a 4–dimensional variational data assimilation system into the Lorenz system; the assimilation scheme was able to reconstruct the trajectory from noisy observations and prevent the nonlinearity of the system to cause the trajectory to diverge too far from the true trajectory.

Furthermore I discussed the theoretical background of variational data assimilation and showed

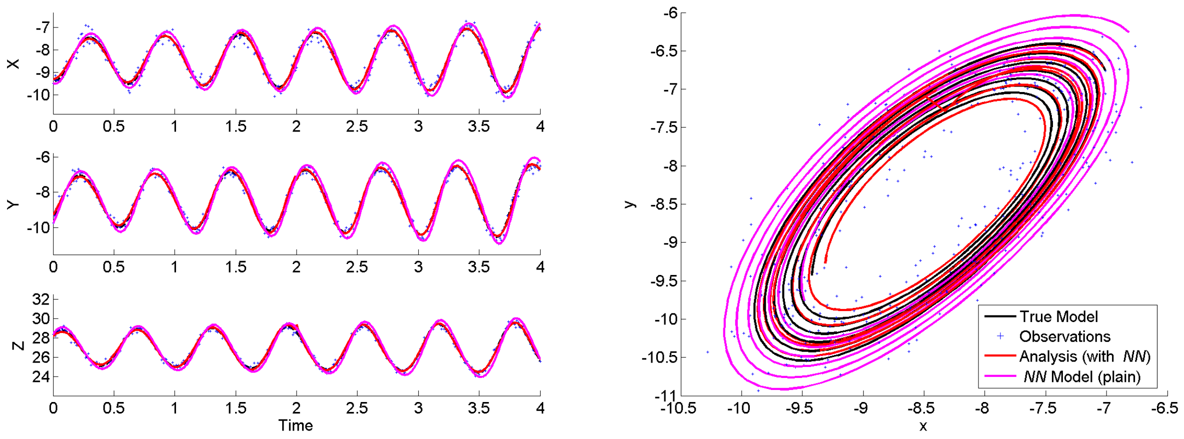

Figure 7: A 4D-Var analysis trajectory of a hybrid model where booth the initial conditions and the network parameter were subject to the optimization algorithm. Annotation and organization as in 6

how to derive a form which also allows for the estimation of parameters within the data assimilation scheme. I also outlined the conceptual similarities between neural network training algorithms and variational data assimilation: essentially both are optimization problems where a cost or error function has to be minimized.

In the main part of this work, following *Tang and Hsieh* (2001), I exploited these similarities and integrated neural network training into the 4–dimensional data assimilation system. The neural network performance did improve once neural network parameters were included in the variational data assimilation scheme, however, not as significantly, as reported by *Tang and Hsieh* (2001). Most notably, in my experiments the instability exhibited by the hybrid model was not as sever as reported by *Tang and Hsieh* (2001).

In all stages one problem persisted: the poor convergence of the optimization algorithms severely limited the success of both, neural network training and variational data assimilation. This is largely a consequence of the highly nonlinear and chaotic dynamics of the Lorenz system; therefore I limited the experiments to a region of only weakly nonlinear behavior.

The initial motivation to integrate neural network training into variational data assimilation was to account for the dynamical constraints of the system in the network training procedure (cf. *Tang and Hsieh*, 2001). The hope was that the instability could be reduced when the network was to be made "aware" of the nonlinear dynamics during the training process.

The results from my experiments, however, do not necessarily support this view.

Instability of the hybrid model was not a problem in the vicinity of the training data; at the same time several factors indicate that the neural networks are not over–fitting the training data (e.g. validation), so that I conclude, that the Lorenz system is simply too complex to be approximated sufficiently well by a network with only 5 hidden nodes. A more complex network architecture and more diverse training data would be more promising approaches for improving the hybrid models forecast skill of the Lorenz model.

This however would only be an exercise in demonstrating the power of neural networks and can hardly be the purpose here.

In my view the power of the integrated approach lies in the control of errors caused by insufficient complexity in the neural network approximation. The data assimilation algorithm can modify the network weights during run time; quantities associated with a large uncertainty can then be adjusted dynamically within their error margins so as to produce the best agreement with observations. Obviously the value of such a “soft parameter” approach for forecasting applications will be rather limited, but there may be an application in reanalysis projects. Furthermore, by monitoring the weight adjustments during runtime, the estimated value of the parameter may be further constrained (similar to the study of analysis increments to identify model biases).

In this sense, I want to abandon the view that the hybrid model (at least the one used here) can be trained to approximate the Lorenz model sufficiently well. Hence there is also little use in giving forecast skills for the hybrid model, for it is not meant to forecast. The (necessary) adjustments due to the variational data assimilation should be viewed as transient corrections or fitting of momentary fluctuations which are not represented by the model; there is evidence in my experiments that over–fitting of neural network parameters does already occur during successful data assimilation runs.

Using variational data assimilation to circumvent nonlinear instabilities, however, will pose a challenge for the robustness of the optimization algorithm. The optimization algorithms used in this study are of a rather primitive nature. I expect more sophisticated algorithms to improve the effectiveness of variational data assimilation. And by the same token I also want to suggest, that modern methods of neural network training should be integrated into the variational assimilation system directly. In this study the dynamical constraint was implemented into neural network training in a rather primitive way and then compared to the most sophisticated conventional algorithms available.

Finally I want to add a remark on the state space in this study and in applications to meteorology and oceanography: in this study the state space was very limited in size (three components) while the state space of current NWP models is approximately seven orders of magnitude larger. Here the number of parameters of the neural network was considerable, compared to the initial conditions. This will probably not be the case in meteorological or oceanographic applications, provided a sensible filtering technique is used and the neural network is not used to predict the whole state space.

Note I am aware that this study is very weak in terms of quantitative support for the given interpretations. This is due to the time constraints under which it was written: systematically acquiring data and casting it into a form that can be published is simply beyond my time frame. Considering this, the present work is probably best seen as a technical outline of the implementation along with some preliminary results and speculations.

References

Bouttier, F., and P. Courtier (1999), Data assimilation concepts and methods.

Cybenko, G. (1989), Approximation by superposition of a sigmoidal function, *Math. control, Signal, Syst.*, *2*, 303–314.

Daley, R. (1991), *Atmospheric Data Analysis*, Cambridge University Press.

Demuth, H., M. Beale, and M. Hagan (2008), *Neural Network Toolbox.*

Hsieh, W. W., and B. Tang (1998), Applying Neural Network Models to Prediction and Data Analysis in Meteorology and Oceanography, *Bull. Amer. Meteor. Soc.*, *79*(9), 1855–1870.

Lu, J., and W. W. Hsieh (1998), On determining initial conditions and parameters in a simple coupled atmosphere–ocean model by adjoint data assimilation, *TELLUS*, *50*(A), 534–544.

Monahan, A. H. (2000), Nonlinear Principal Component Analysis by Neural Networks: Theory and Application to the Lorenz System, *J. Climate*, *13*, 821–835.

Polavarapu, S. (2008), Data Assimilation, lecture notes.

Tang, Y., and W. W. Hsieh (2001), Coupling Neural Networks to Incomplete Dynamical Systems via Variational Data Assimilations, *Mon. Wea. Rev.*, *129*, 818–834.