Seminar MPOC:
The use of Neural Networks in Ocean Modelling

Erik van Sebille

June 6, 2005

Abstract

This paper gives a short introduction in the use of Neural Networks in ocean modelling. It shows that the mathematical foundations of neural networks, both in a topological and a mathematical/algorithmic sense are firm. There are a lot of questions that have to be dealt with, but in principle the technique is adequate.Some comments are given on the process of training an Neural Network. This is a complicated but crucial part in the successful use of Neural Networks in modelling in general.

Finally, four examples of the use of Neural Networks in ocean modelling are discussed.

(1) A Neural Network approach to solve the UNESCO equation of state, cutting computational effort on this part of the large ocean models. The technique can also be applied to do the inverse: calculate salinity from density and temperature.

(2) A Neural Network approach to model nonlinear wave interactions. This problem is far more complex and the final solution is still far away. However, progress has been made.

(3) A Neural Network approach to the principal component time series analysis. The nonlinear nature of Neural Networks offers great advantage, especially when analyzing oscillatory or otherwise nonlinear processes. This is backed by the improvement in the analysis of the ENSO phenomenon.

(4) A Neural Network approach to translate remote sensing data to geophysical variables. The Neural Network is used to process SSM/I data in a very accurate, fast and easy way. This example does not show a particularly intricate Neural Network, but it accentuates the diversity in capabilities of simple Neural Networks.

1 Introduction

Neural Networks is a technique emerging as a tool for ocean modelers. It's use has been widespread under the scientific community for a while, and since the late nineties it is also implemented in geophysical models. Most applications are in the meteorological field of research, but can be extended to the ocean. In this

paper, the Neural Network-method will be discussed from the ocean modelling point-of-view. First, a theoretical description of Neural Networks is given, based on the paper by Krasnopolsky et al. [2002]. After this, four examples of Neural Networks in ocean modelling will be discussed. They will be:

- The equation of state, following the paper by Krasnopolsky et al. [2002].
- The wind wave interactions, after the paper by Tolman et al. [2005].
- Principal component time series analysis, based on the papers by Hsieh [2004] and by Hsieh and Tang [1998].
- Remote sensing data processing, following the paper by Krasnopolsky and Schiller [2003]

The paper ends with a discussion on the problems and promises of Neural Network use in ocean modelling.

1.1 A brief history

Hsieh and Tang [1998] give a nice overview of the development of Neural Networks in general. The Neural Network technique originated when biologists wanted to develop a model for the human brain in the 1940's. They were fascinated by the complexity and abilities (memory, speech recognition) of the brain and tried to mimic these abilities by making very complex graphs. They hoped that in this way they could give these graphs some self-learning capabilities.

Years passed and the final breakthrough came with the rediscovery of the back-propagation algorithm in 1986 by Rumelhart et al. [1986]. Through this algorithm (see also section 2.2), Neural Networks could be trained in a time-effective manner.

After this breakthrough, scientists became interested to use Neural Networks for applications in many different fields of research. Neural Networks are now used in the field of biometrics, where the need for pattern recognition is imminent. Neural Networks can be trained for e.g. speech, facial, iris and fingerprint recognition.

Financial forecasters also make use of Neural Networks when trying to predict stock rates. They use Neural Networks as black boxes for analyzing time series without any assumptions on the forces and mechanisms that control the economy.

Neural Networks are also used in bio-informatics to relate DNA to gen-activity. And finally, meteorologists use Neural Networks to distinguish different kinds of clouds in satellite images.

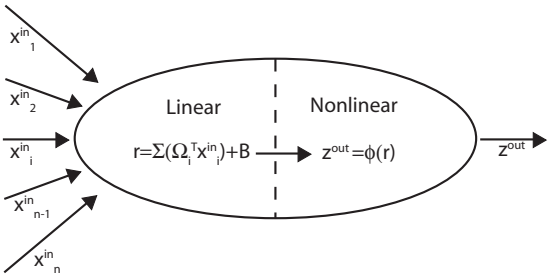

Figure 1: A model of a neuron, consisting of a linear and a nonlinear part. After Krasnopolsky et al. [2002]

2 Theory

2.1 Creating a Neural Network

There are two ways to look at a Neural Network. The first one is the topological, in which the network is presented as a directed graph with nodes and edges, representing the flow of data through the network. The other is a more mathematical/algorithmic point of view. The Neural Network is a set of consecutive sums of weighted input variables. Both views have their advantages and disadvantages, and they will be both be used throughout this paper.

A Neural Network constitutes a mapping F of an input vector X with parameters $X = \{x_1, x_2, \ldots, x_n\}$ to an output vector $Y = \{y_1, y_2, \ldots, y_m\}$. The mapping is continuous except for a finite number of discontinuities:

$$Y = F(X), \qquad X \in \mathbb{R}^n, \quad Y \in \mathbb{R}^m \tag{1}$$

The actual mapping is done by a network of neurons or nodes. These neurons are the processing elements. They require an input vector X^{in} and return a value z^{out}. A typical neuron is shown in figure 1. The neurons consist of a linear part and a nonlinear part. The linear part returns the sum r of the inner product of the vector X with weights Ω and a bias B. The result r, a linear combination of the input vector, is the argument for some nonlinear function ϕ, which returns the value z^{out}. ϕ is called the activation function. It can be written as

$$z^{out} = \phi\left(\sum_{i=1}^{n} \Omega_i^T x_i^{in}\right) + B \tag{2}$$

Any nonlinear function will do for ϕ but common choices are the logistic function $\phi = \frac{1}{1-e^{-r}}$ and the hyperbolic tangent function $\phi = \tanh(r)$. Both function stay bounded as $x \to \pm\infty$, making sure that the weights Ω_i will not become extremely large. Moreover, the derivatives of these functions are easily computed. This is required for the back-propagation teaching algorithm (see

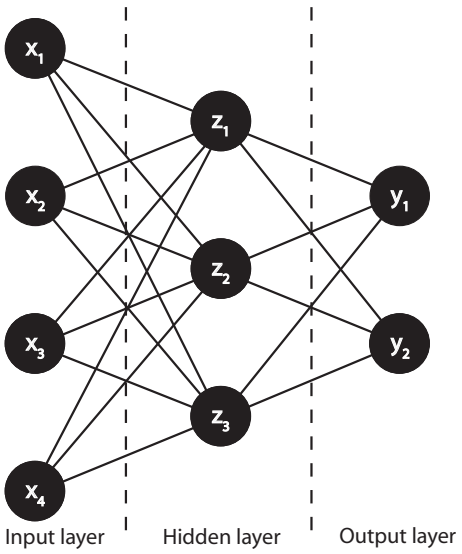

Figure 2: The Neural Network, made up by several bipartite connected layers of neurons. In this case an input, a hidden and an output layer. The nodes are neurons as depicted in figure 1

section 2.2). Note that a neuron can easily be made linear by choosing the identity, $\phi = r$, for the activation function.

A neuron has an input vector and an output scalar and is contained in a large graph of neurons, the Neural Network (see figure 2). This graph is directed and consists of several layers that are bipartite connected. It has a feed-forward direction, from the input via the hidden to the output layer. In this form, it is referred to as the Multi-Layer Perceptron (MLP). The neurons in the input layer are linear. They simply distribute the vector X to the hidden layer. In this layer the neurons are nonlinear as depicted in figure 1. The nodes in the output layer, can be either linear or nonlinear and are the collectors for the vector Y.

The number of neurons in the input layer is equal to the dimension of the vector X, the number in the output layer to that of the vector Y. One is free to choose the number of nodes in the hidden layer. It should depend on the complexity of the problem to solve and will require some trial-and-error.

We can now write down an expression for a full Neural Network with one hidden layer, as depicted in figure 2. We assume that the first layer only collects the separate elements of the vector X, i.e.

$$\Omega = I, \quad B = 0 \qquad \forall \quad \text{neurons in input layer} \tag{3}$$

with I the identity matrix. We also assume that the activation function for the output layer is the identity, $\phi = r$.

In this configuration, the expression for an element y_q of vector Y as a function of the input elements x_i of vector X and an activation function $\phi = \tanh(r)$ can be written as

$$y_q = \sum_{j=1}^{k} \omega_{qj}^T \left[\tanh \left(\sum_{i=1}^{n} \Omega_{ji}^T x_i + B_j \right) \right] + \beta_q \tag{4}$$

where Ω_{ji} and B_j are the weights and biases of the neurons in the hidden layer and ω_{qj} and β_q are the weights and biases in the output layer.

Using equation (4), the implementation of a Neural Network in a computer language is straightforward and requires only additions and multiplications, provided that the language has a good and fast hyperbolic tangent function. Most computer languages do.

It is also possible to acquire the Jacobian of the mapping, an array of all possible partial differentials $\frac{\partial y_q}{\partial x_p}$, with little computational burden. The partial differentials can be written as

$$\frac{\partial y_q}{\partial x_p} = \sum_{j=1}^{k} \left(1 - \left[\tanh \left(\sum_{i=1}^{n} \Omega_{ji}^T x_i + B_j \right) \right]^2 \right) \Omega_{pj} \omega_{jq} \tag{5}$$

and is in this notation not very different from equation (4). This Jacobian is very useful in some applications, such as the computation of the equation of state for seawater (see section 3.1).

2.2 Teaching a Neural Network

Before the Neural Network can be used to map the input vector X to the output vector Y, the parameters Ω_{ji}, B_j, ω_{qj} and β_q in equation (4) need to be known. The technique to obtain these parameters is called teaching. The idea of teaching is to show the Neural Network a set of (X, Y)-pairs. After it has learned the mapping between those pairs, it should be able to construct the output vector of a never encountered input vector. It is because of this teaching, that people often talk about artificial intelligence in the context of Neural Networks. However, it is nothing more than smart interpolation and curve-fitting between already known (X, Y)-pairs. There are numerous techniques that do this, the only advantage of the Neural Network technique is that it uses nonlinear interpolations.

To obtain the parameters Ω_{ji}, B_j, ω_{qj} and β_q in equation (4), the Neural Network needs to be fed with a practise set. This practice set is usually an already known subset (X_d, Y_d) of the mapping that is researched. By feeding the subset to the Neural Network, the weights are set to appropriately map the input X to the output Y. The technique can therefore only be used when both a set X_d and a corresponding set Y_d are at hand.

The most renowned algorithm for teaching a Neural Network is the back-propagation algorithm by Rumelhart et al. [1986]. The algorithm iteratively

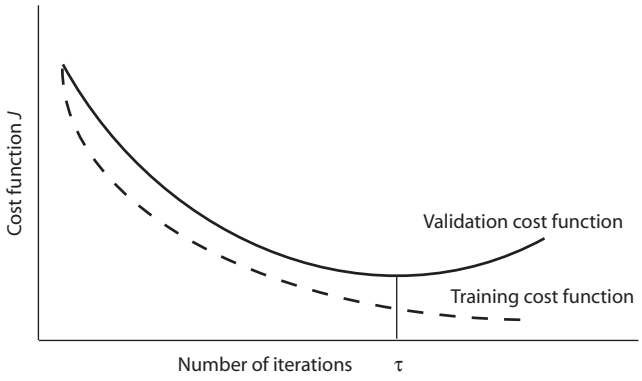

Figure 3: The cost function of the training set (dashed curve) and the validation set (solid curve) as a function of the number of iterations of the training algorithm. The minimum of the validation cost function at iteration step τ is the time to stop the training. (Figure after Gardner and Dorling [1998])

searches for a global minimum of the cost function

$$J = \sum_{q=1}^{m} (y_q - y_{dq})^2 \tag{6}$$

where y_q are the elements of the output vector Y and y_{dq} are the corresponding elements of the training set vector Y_d. It is difficult to find a global minimum, as the function J can be very nonlinear and have many local minima. The algorithm can easily get stuck in such a local minimum and an observer will never be sure that the reached minimum is the global minimum. Enlarging the practise set and increasing the number of iterations in the back-propagation algorithm are techniques to decrease the chance of a reached minimum not being the global minimum.

However, the number of iterations should not get too large. If this is the case, there is a possibility of overfitting, especially if the data contains noise. The algorithm will try to fit to all the noise and it will not behave optimal on a new input vector.

One way to solve the problem of overfitting is to have another data set, the validation set, at hand. During the training period, the network is trained on the practise set and its performance is checked on the validation set. When overfitting starts to occur, the cost function of the validation set will start to get larger (see figure 3). This is the time to stop.

The selection of the training set (X_d, Y_d) also requires some attention. Especially when the teaching subsets are small or misbalanced compared to the expected input on operation, good results are not guaranteed. Misbalanced means that the subset is not a good representation of the total set. Neural Networks are better in interpolating than in extrapolating so attention has to

be paid to the selection of the subset. Gardner and Dorling [1998] note that especially for prediction, scientists are often interested in extremes (thunderstorms, floods). Such events by definition occur very rarely and a training set made up by these events will be small. One can use a teaching set built from normal weather conditions, but this requires extrapolating when interested in the extremes, limiting the use of Neural Networks in such cases.

In the remainder of this section, three techniques shall briefly be discussed to improve the training mechanism. Their actual usefulness may be questioned, but in general they give extra insight in the complicated process of training a Neural Network.

2.2.1 Validity checks

It is possible to signal occurrences of extrapolation. Krasnopolsky and Schiller [2003] note that if both the mapping between vectors X and Y and the inverse mapping F_{inv} exist, another Neural Network can be built with X and Y swapped. Now, by definition:

$$X = F_{inv}(F(X)) \tag{7}$$

if the mappings are correct. This is normally the case in the subspace where the Neural Network was tought to interpolate. In subspaces of X where the mapping was destined to extrapolate however, this equation (7) is not in general true because of the inferiority of the mappings.

Therefor, during the execution of a Neural Network, the occurrence of a data set outside the training set can be signaled using a validity check

$$\sum(X - F_{inv}(F(X))) > \varepsilon \tag{8}$$

with ε some threshold value. When equation (8) is true, the operator can be warned and must handle appropriately.

2.2.2 Weight penalties

Hsieh [2001] discusses a different approach to the problem of overfitting. He argues that overfitting is very often caused by an exaggeration of the non-linearity of the problem at hand. This would especially be the case when noise is not a factor.

The activation function $\phi = \tanh(r)$ with $r \sim \Omega \cdot X^{in}$, has the property that for a given X in the bounded interval $[-L, L]$, it can go two ways. If Ω is very small, $\phi \sim X$, being almost linear. If Ω is very large on the other hand, ϕ will approach the extremely non-linear step function. Hsieh states that this is unwanted behavior.

To avoid the last from happening, Hsieh introduces the notion of a weight penalty. This is a term added to the cost function making sure that the weights Ω_{ji} will not become too large:

$$J = \sum_{q=1}^{m} (y_q - y_{dq})^2 + p \sum_{j=1}^{k} \sum_{i=1}^{k} \Omega_{ji}^2 \tag{9}$$

with p the weight penalty parameter.

This weight penalty parameter must be tuned as a trade-off between a less non-linear solution and an over-linearized one. This makes the procedure less attractive, but it is good to mention it as a way to solve overfitting of noise-free data sets.

2.2.3 Sobolev normed cost functions

Krasnopolsky and Chevallier [2003] comment on the cost function J being used. They note that when the main goal for the Neural Network is to produce the Jacobian (equation (5)), it is recommended to use an alternative cost function. During the training period the Jacobian is not taken into account and therefor its accuracy during operation may not be optimal. One could off course include the Jacobian in the training set, but this increases the complexity both of the training set and of the Neural Network. Another approach is to build a second Neural Network dedicated to compute only the Jacobian, but this doubles the effort spent both on training and during operation.

The best way to include the Jacobian in the training process is to alter the cost sum, which is just an Euclidian norm, to a more general form. In a so called Sobolev norm (a generalization of the Euclidian norm), the derivatives of a function are also taken into account. This implies that if the training set vector Y_d approaches Y, but also gets more oscillatory, the cost function will not generally decrease. This is because the 'distance' between the derivatives increases.

The new cost function can now be written as

$$J = \sum_{q=1}^{m} (y_q - y_{dq})^2 + \sum_{q=1}^{m} \sum_{i=1}^{k} \left(\frac{\partial y_q}{\partial x_i} - \frac{\partial y_{dq}}{\partial x_i} \right)^2 \tag{10}$$

where y_q are the elements of the output vector Y and y_{dq} are the corresponding elements of the training set vector Y_d.

The drawback of using this new cost function is that training time will significantly increase due to a higher complexity of J. Moreover, it is possible that to accomodate a reasonable value for J, the number of hidden neurons has to be increased, at the cost of more computational effort during execution.

3 Examples

3.1 The equation of state

Large oceanographic models used for climate study are very time-consuming in their calculation of the evolution of the state of the ocean. Runs can take up to months for the entire ocean. The models basically perform two kinds of calculations during these runs. On the one hand, there is the time-integration of the governing differential equations, simplifications of the Navier-Stokes equations. Code for this is often highly optimized.

On the other hand, models must perform complicated mathematical equations. One of these is the equation of state for seawater, relating temperature, salinity and pressure to a density. This density is required as a parameter by the governing equations. In some cases, calculation of the equation of state can take up to 20% of the total computation time (Griffies et al. [2000]). Improving this calculation scheme can reduce computation time significantly (remember that these models typically run for weeks, and a 10% increase in speed can reduce that by days).

The relation between temperature, salt, pressure and density are governed by the Unesco Equation of State (UES). The UES is an empirical relation agreed upon in 1981. This relation is stated in Gill [1982]. The actual equation consists of a number of consecutive steps with 40 parameters:

1. First the density $\rho(0, T, 0)$ of pure water for a given temperature is calculated using a polynomial of degree five in the temperature.

2. Two additional polynomials of degree four are multiplied by the salinity to obtain the density at one standard atmosphere, $\rho(S, T, 0)$.

3. The above two steps are repeated to obtain a bulk modulus $K(S, T, 0)$.

4. Calculation of the bulk modulus $K(S, T, p)$ requires solving a polynomial with fourteen different terms.

5. Finally, the density at pressure p is

$$\rho(S, T, p) = \frac{\rho(S, T, 0)}{1 - \frac{p}{K(S,T,p)}} \tag{11}$$

Because of the 40 parameters, direct calculation of the UES is cumbersome to say the least. And because of the dependence on three fields (the 3-dimensionality) an *a priory* calculation for the density stored into a lookup table takes too much memory. Building such a table with a range of $D = \{-2^\circ < T < 40^\circ\text{C},\ 0 < S < 40 \text{ psu},\ 0 < P < 10,000 \text{ decibar}\}$ and precision $\sigma = \{T \pm 0.01^\circ\text{C},\ S \pm 0.01 \text{ psu},\ P \pm 0.1 \text{ decibar}\}$ would require a table with $1.7 \cdot 10^{12}$ elements. This is an impossible number, even for supercomputers.

Krasnopolsky et al. [2002] describe how the problem of solving the equation of state for seawater can be addressed in a Neural Network setting. They created

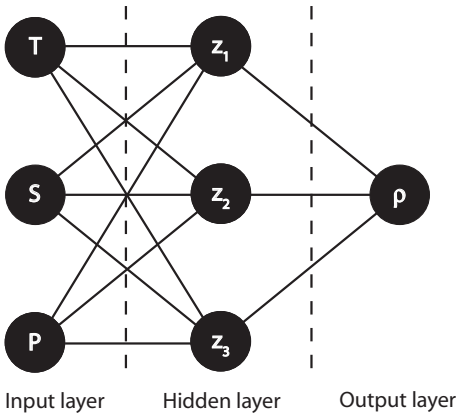

Figure 4: The Neural Network for solving the UNESCO Equation of State. The nodes z_k in the hidden layer are neurons as depicted in figure 1

a Neural Network with three neurons in the input layer (representing the input $\{T, S, p\}$), three in the hidden layer and one in the output layer, representing the output ρ. The total network is depicted in figure 4.

Using this graph, calculating the density via equation (4) consists of a total of 12 multiplications, 25 additions and taking 3 tangent hyperbolics (see the algorithm 1). This is far less than the calculation needed for the original UES.

Algorithm 1 An implementation of the UES Neural Network algorithm

$x = \{T, S, P\}$
$\rho = 0$
for j=1 to 3 **do**
  $r = 0$
  for i=1 to 3 **do**
    $r = r + \Omega_{ji} \cdot x_i$
    $r = r + B_j$
  end for
  $r = \tanh(r)$
  $\rho = \rho + \omega_j \cdot r$
end for
$\rho = \rho + \beta$
return ρ

Off course, when reforming a deterministic function like the UES to an approximate function like equation (4), errors are bound to be made. However, due to variations in the composition of the dissolved salts in the sea water and uncertainty in the pressure, the uncertainty in the UES is about 0.05–0.1 kg m^{-3}. So if the Neural Network algorithm stays within an error of 0.1 kg m^{-3},

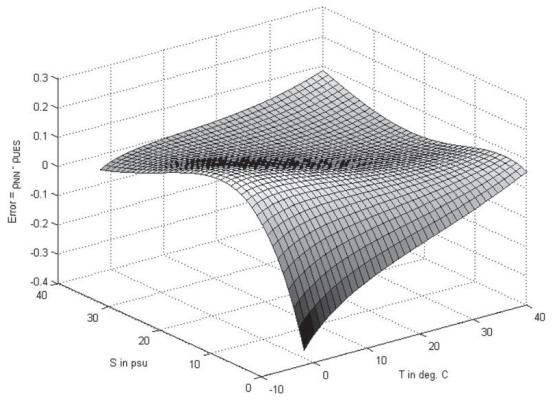

Figure 5: Error of the Neural Network algorithm as a function of temperature T and salinity S at the ocean surface ($z = 0$). Figure from Krasnopolsky et al. [2002]

this method is not worse than applying the UES directly.

Krasnopolsky et al. [2002] constructed a teaching set of 4000 points in the $\{T, S, p\}$ domain. Because the corresponding densities are exactly known (at least, the UES outcome is), they could easily evaluate the cost function equation (6). Moreover, the entire scope of expected values can be dealt with without having to deal with noisy data so a lot of the problems discussed in section 2.2 do not apply to this case.

This resulted in a configuration of the Neural Network parameters Ω_{ji}, B_j, ω_j and β such that the error in the density as calculated by the Neural Network is shown in figure 5. It shows that the calculated density is never off more than 0.1 kg m^{-3} except at the combination of low temperature and low salinity, which is never encountered in the ocean.

In principle, it is odd that the authors even find such a large error. Their training set can comprise the entire domain, and can be as large as they need it to be. That is the advantage of modelling a deterministic equation. However, if this is the case, errors should be negligible. The authors can always decide to add more $\{S, T, p, \rho\}$ pairs of the UES to the training set and this will always improve the neural network.

The authors do not comment on this, but it is likely that the number of hidden neurons will be the bottleneck in this case. With only three hidden neurons, the degrees of freedom of the neural network will be limited. It would be interesting to see whether adding a hidden neuron would decrease the errors (at a trade-of with increasing computational effort).

However, we can conclude that the Neural Network approach to compute the UES is within the error of the UES itself, at least for physically expected values. The authors do not mention how their method does in the deep ocean, but the results they show are certainly hopeful.

The authors remark that the density field does not have to be calculated for

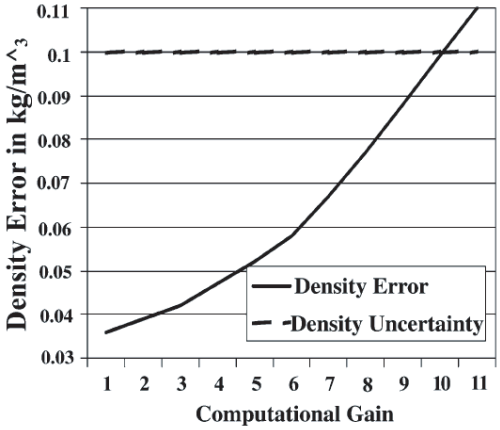

Figure 6: Error in the density as a function of computational gain based on the use of equation (12). Figure from Krasnopolsky et al. [2002]

every time step. One can also calculate the total differential

$$\Delta\rho = \frac{\partial\rho}{\partial T}\Delta T + \frac{\partial\rho}{\partial S}\Delta S \tag{12}$$

where ΔT and ΔS are small differences in the temperature and salinity resulting in a small difference in the density. The two partial derivatives $\frac{\partial\rho}{\partial T}$ and $\frac{\partial\rho}{\partial S}$ can easily be obtained from the neural network by equation (5).

Assuming that the time step is small compared to the advective time scale, the partial derivatives are valid for several tens of time steps. During these time steps, the density can be calculated at virtually no computational cost. After these time steps, the derivatives are recalculated. The authors claim almost no extra errors and a nine times faster computation using this technique. This is shown in figure 6.

We can conclude that the Neural Network technique is a promising technique to apply to the UES. It vastly reduces computational effort and brings no extra errors above the errors already present in the UES.

3.1.1 Retrieving Salinity

Besides discussing the usefulness of Neural Networks to approximate the UNESCO equation of state (UES), Krasnopolsky et al. [2002] also discuss a neural network application to the inverse problem, the calculation of salinity.

Ocean models often include the data assimilation of Sea Surface Temperature (SST) retrieved from satellite observations. These observations are then used as a boundary condition for the model at the ocean surface. This implies changing the temperature in the upper layer of the ocean. But when the temperature is

changed, one should be careful to avoid gravitational instability. Decreasing the SST without decreasing the salinity can result in an unstable water column.

The easiest way to solve this is to adjust the salinity to a value that gives a marginally stable water column. In this way, the salinity field has to be altered the least and stability is still preserved.

The question is therefor how to calculate the salinity given the temperature and the density. This requires inverting the UES, an even more cumbersome calculation than the one presented in the previous section, involving iterating to the desired value for the density. However, since the relation between the three variables is known, we can use a Neural Network approach. As we are only correcting for the ocean surface, pressure influence can be ignored. The input vector is made up by $\{\rho, T\}$ and the output vector by S. Building a training set is straightforward, using the normal UES.

Again the authors claim success, as the error in salinity (when related to a density error) does not exceed the 0.1 kg m^{-3}.

3.2 Wind wave interactions

Another field in ocean modelling where the use of Neural Networks is emerging is that of wind wave interactions. However, unlike the equation of state, the Neural Networks approach has not matured here. Results are promising but there is no final model of how to effectively produce a Neural Networks method for this field of research. One reason for this is that the problem at hand is far more complex than the equation of state.

Scientists interested in the prediction of the wave field use the conservation equation (following Tolman et al. [2005])

$$\frac{dF}{dt} = S_{tot} = S_{in} + S_{nl} + S_{ds} \tag{13}$$

where the left-hand side denotes the evolution of the wind spectrum over time, S_{in} is the source term of wind, S_{nl} represents the non-linear interactions and S_{ds} is the dissipation of the wind spectrum.

The non-linear interaction is the term we are interested in for the moment. It represents the lowest order mechanism to shift wave energy from short to longer waves, from the main input scale to the dissipative scales. It thereby maintains and stabilizes the shape of the spectrum.

The non-linear interaction can be described as the resonant exchange of energy, momentum and actions between four components in the wave spectrum, in such a way that

$$\begin{aligned} \vec{k}_1 + \vec{k}_2 &= \vec{k}_3 + \vec{k}_4 \\ \sigma_1 + \sigma_2 &= \sigma_3 + \sigma_4 \end{aligned} \tag{14}$$

for wave numbers $\vec{k}$ and wave frequency σ.

If the action spectrum is defined as $n \equiv F/\sigma$, the rate of change of this action spectrum is expressed as

$$\frac{\partial n_1}{\partial t} = \int \mathrm{d}\vec{k}_2 \int \mathrm{d}\vec{k}_3 \int \mathrm{d}\vec{k}_4 \; G(\vec{k}_1, \vec{k}_2, \vec{k}_3, \vec{k}_4)\delta_k\delta_\sigma[n_1 n_3(n_4 - n_2) + n_2 n_4(n_3 - n_1)] \tag{15}$$

in which $n_i = F(\vec{k}_i)/\sigma_i$ and G is a coupling coefficient.

This equation (15) involves a six dimensional integration (as $\vec{k}$ is two dimensional) and is therefor very time-consuming. This gets even worse as G is a complicated, highly non-linear function that contains singularities.

Implementation of this wave action equation into wave models makes computing S_{nl} take up approximately 10^4 times more computational time than the rest of the terms in equation (13). Therefor, improvements in the computation of S_{nl} directly result in a drop in computational effort for wave prediction.

There already exists a method to decrease the computational effort in the non-linear interaction term. It is called Discrete Interaction Approximation (DIA, Hasselman et al. [1985]) and evaluates the six integrals in equation (15) using only a few discrete values for the wave numbers $\vec{k}$. This DIA is now used in wave models, but has some shortcomings. Most importantly, the higher frequencies suffer from large systematic errors that are not yet overcome.

Because of the problems with the DIA, a Neural Network approach is now tried. In this approach, the non-linear term S_{nl} is mapped on the spectrum F

$$S_{nl} = T(F) \tag{16}$$

This mapping has very high dimensionality because of the range of the two vectors S_{nl} and F. Krasnopolsky et al. [2002] even speak of vectors with over 600 elements (neurons). The hidden layer should have at least the same order or neurons, giving the equation (4) a huge degree of freedom. This is unpractical, since it will take much of the computational time to solve this equation.

In order to reduce the number of neurons for the two vectors, the spectrum F and interaction term S_{nl} are represented by two sets of orthogonal functions Φ_i and Ψ_i.

$$F \approx \sum_{i=1}^{n} x_i \Phi_i \quad \text{and} \quad S_{nl} \approx \sum_{i=1}^{m} y_i \Psi_i \tag{17}$$

In this way the dimensionality can be controlled, at the cost of creating systematic errors, not to rise exorbitantly. However, n and m can always be chosen in such a way that there is no loss of generality.

The x_i and y_i in equation (17) are the input and output of the Neural Network, equation (4). In this way, mapping a spectrum to the non-linear term in equation (13) requires three basic steps:

1. Decompose the spectrum F to a vector X

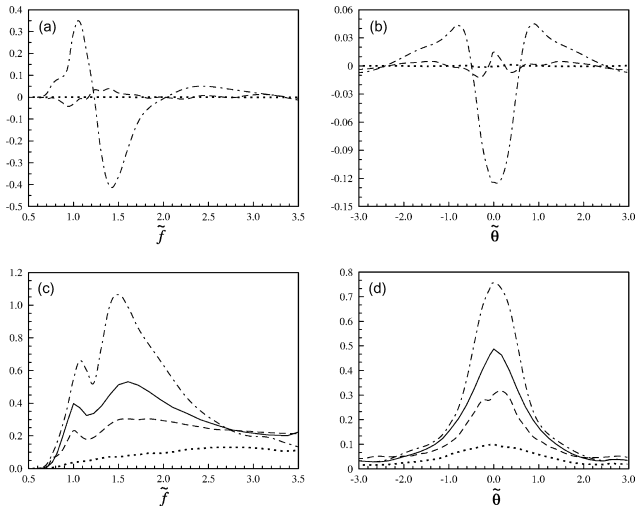

Figure 7: One dimensional biases (upper panels) and rms errors (lower panels) as a function of $\tilde{f}$ (left panels) and direction $\tilde{\theta}$ (right panels) for the DIA (chain line) and two versions of the Neural Network implementation (dashed and dotted line). The solid line is not discussed in this paper. Figure from Tolman et al. [2005]

2. Feed the vector X to the Neural Network to obtain Y

3. Compose S_{nl} from Y

This enumeration of steps already reveals one of the disadvantages of this approach. The composition and decomposition, steps 1 and 3, take up a lot of time. This is one of the points that can be improved in the future.

On overall, Neural Networks is a promising technique to calculate the non-linear interaction from a wave spectrum. Tolman et al. [2005] find an order of magnitude improvement in some special cases against the DIA model. This is depicted in figure 7, where the authors plotted the errors in direction and frequency of a validation set of 10,000 spectra. The authors use two different functions for Φ and Ψ (equation (17)) and compared these with the DIA model. Both functions performed better on the validation set, giving good hope that this method can be a useful tool for modelling wind wave interactions when it has fully been developed.

3.3 Time series analysis

A third example of the use of Neural Networks is for performing time series analysis. Time series analysis is not widely used in ocean modelling, but it is common practice in the field of climatology. Since there are many connections

between ocean modelling and climatology, it is also useful do discuss results of Neural Network implementations in the time series analysis.

Hsieh [2004] discusses how to use Neural Network methods in the time series analysis. In the time series analysis, one tries to make predictions about the future based on knowledge about the past. The use of Neural Networks for these kind of problems is widely supported. As stated in the introduction, even stock brokers make use of Neural Network techniques when trying to outsmart the market.

The advantage of Neural Networks is that no *a priori* assumptions have to be made about the relations between variables. Moreover, these relations do not have to be linear, as was the case with previous methods to analyze time series.

The method discussed here is the Principal Component Analysis (PCA). This method iteratively finds the principal components (linear coefficients) that make up a time series.

The method starts by finding a linear combination u of a time series $f(\vec{x}, t)$ and associated spatial vector $a(\vec{x})$, with

$$u(t) = \sum_{i=1}^{n} a_i(\vec{x}) f(t, \vec{x}) \tag{18}$$

in a way that the cost function

$$J = \sum (f(\vec{x}, t) - a(\vec{x})u(t))^2 \tag{19}$$

is minimized. The associated vector a is the coefficient of the first mode, as it is called, of the PCA. When it is found, the procedure is repeated with the residual $(f(\vec{x}, t) - a(\vec{x})u(t))$ to retrieve higher modes, corresponding to higher polynomials.

Linear methods such as the PCA are known to poorly resolve oscillatory signals, such as seasonal variability. These processes are inherently non-linear and should therefore require a non-linear analysis. The Neural Network method as described in section 2 is a non-linear method and is therefor expected to perform better on oscillatory problems.

A Neural Network used for time series analysis looks somewhat different from the graphs we have already encountered. Instead of a linear map through the vector a in equation (18), a nonlinear mapping

$$u(t) = g(X(t)) \tag{20}$$

is allowed for. To do this a graph with three hidden layers is constructed (see figure 8). The middle of these is the bottleneck layer and consists of only one neuron. It is this neuron that performs the nonlinear map of equation (20). The other hidden layers are used to transform between the observables and the bottleneck. These layers are appreciated if the total Neural Network in figure 8 is viewed upon as two Neural Networks glued together. One maps from X

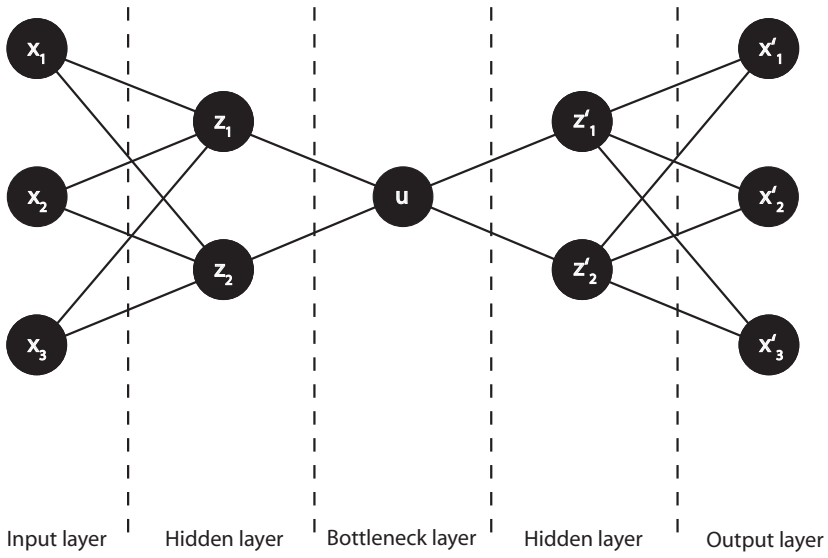

Figure 8: The Neural Network for the time series analysis. The hidden layers are used to transform between input vectors X and X' to a dimension acceptable for the bottleneck neuron u. It is this bottleneck neuron u (always exactly one) that computes the principal components. Higher modes can be obtained by iteratively feeding the network with residuals.

to u and the other maps from u to X'. Both networks need a hidden layer to fully make use of the potential of the Neural Networking method in that all non-linear combinations between input and output variables are possible.

The Neural Network can be trained by supplying a training set made up by many observations $X = \{x_1, x_2, \ldots, x_n\}$ and minimizing the cost function

$$J = \sum_{i=1}^{n} (x_i(t) - x_i'(t))^2 \tag{21}$$

That is, the output vector X' should follow the input vector as close as possible. If done in this way, the bottleneck neuron u gives the nonlinear principle component, equivalent to the modes A of the PCA. The residual of this component (i.e. $x - g(u)$) can pass the network to iteratively give higher modes.

It is important to note that the Neural Network in this case is very different from the previous examples we have seen. Not only is the number of layers extended, the use of the network has also changed. In the previous examples, the Neural Network was first trained with a training set, and then used to process unseen input to output. The exact nature of the weights and biases inside the Neural Network was unimportant, and the hidden layers were in that sense truly hidden.

In the Neural Network PCA, the network is used to obtain the coefficients.

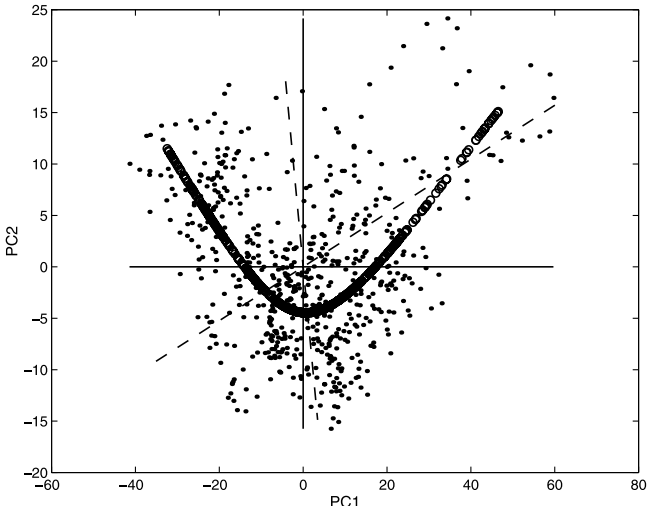

Figure 9: Scatterplot of the sea surface temperature anomaly data (dots) in the principal component plane for the first two modes PC1 and PC2. El Ninõ states are situated in the top right corner, while La Ninã states are in the top left corner. The first mode PCA eigenvector is along the horizontal line, and the second mode eigenvector is along the vertical line. The first mode of the Neural Network PCA is shown as circles. It is clearly non-linear and fits the extremes much better. The meaning of the dashed lines is not discussed in this paper. (From Hsieh [2001])

After training, the network will not be used for processing. Instead, the weight and bias of the bottleneck neuron are retrieved and used for further research. The network has become redundant after its training is completed.

Hsieh [2004] uses this method to analyze the El Ninõ Southern Oscillation (ENSO). He compares a regular PCA with the Neural Network PCA. He finds that when only two principal components are taken into account, the PCA very poorly resolves the extremes of an El Ninõ and La Ninã occurrence (see figure 9).

The Neural Network PCA does much better on these instances. Because of its non-linearity it can, even in only one mode, capture the oscillatory behavior in the ENSO. Moreover, the spacial pattern of the NNPCA for different values of X varies continuously.

It can therefor deal better with the asymmetry in the maximum of El Ninõ and La Ninã situations. This is depicted in figure 10, where the greatest variability in the PCA modes is near the Peruvian coast. This would imply that La Ninã is the mirror of El Ninõ, something not seen in the ocean. The Neural Network does a much better job, situating the minimum of u in the middle of the Pacific, just as observations show.

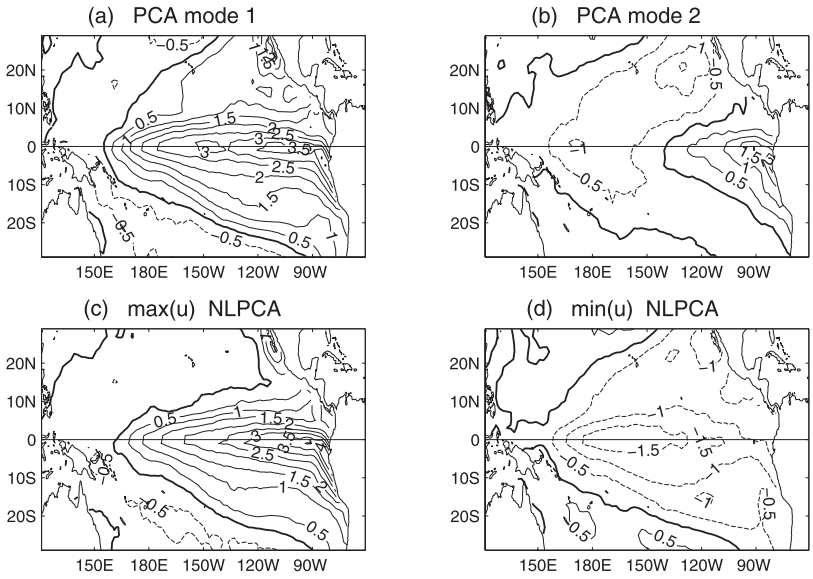

Figure 10: Sea surface temperature anomaly pattern of the PCA (a) and (b) and the Neural Network PCA (c) and (d). The largest spatial variability is near the Peruvian coast in the two PCA modes. In the Neural Network solution, the minimum (La Ninã) is situated in the deep ocean, a much better simulation of the data. (Hsieh [2001])

3.4 Remote sensing

In remote sensing, the problem is that sensor data S has to be converted to geophysical parameters G. The data from the sensors in satellites is often a measurement of radiation in different frequencies and polarization. It is not a straightforward task to extract relevant information from this.

The problem is one of casuality: even if it may be possible to derive a relation between the geophysical parameters G and the sensor data S from physical considerations, it may very well be that the inverse relation can not. The relation between S and G does not correspond to a cause and effect principle; i.e. the satellite data does not influence the geophysical parameters.

However, it is this mapping

$$G = F(S) \tag{22}$$

that we seek when we want to interpret satellite data.

A lot of techniques have been developed to obtain this mapping, most using iterating to a correct G for the known mapping $S = F(G)$. This can be done both linearly and nonlinearly, but in general achievements have not been perfect.

Besides the problems discussed above, the assimilation of remote sensing retrieved data into for instance operational weather prediction models is also

very time relevant. The data assimilation part of a model should not take too much time, better spent on iterating the governing differential equations.

These problems of both an unknown nonlinear continuous dependency of geophysical parameters on remote sensing data and the urge of fast calculation has motivated Krasnopolsky and Schiller [2003] to discuss a Neural Networks approach in remote sensing.

The authors illustrate the advantages of a Neural Network by using data from the SSM/I instrument. This instrument is an array of identical satellites launched through the US defense meteorological satellite program (DMSP). The satellites orbit the earth covering every ocean basin twice a day. The instrument measures radiation in seven channels. At the frequencies of 19, 37 and 85 GHz it senses the horizontal and vertical polarization, at 22 GHz only the vertical polarization.

The SSM/I is used to obtain ocean wind velocity, columnar water vapor, columnar liquid water and sea surface temperature (SST) over the oceans.

The Neural Network used by the authors is a pretty straightforward one. The graph consists of an input vector S of five instrument channels (the 85 GHz channels are omitted), a hidden layer of 12 neurons and an output layer G with the four geophysical parameters.

From buoy and ship observations, the authors constructed a training set. A problem was that these data become sparse at high wind velocities, when ships stay in the harbor and buoys fail to accurately sample wind velocity due to rough conditions. So even if the authors wanted to, they could not construct a balanced training set.

It is therefor that they develop the concept of a validity check (see also section 2.2). In this way, although extreme storms can not be sampled during operation, they can at least be flagged. It is for the operators to decide on appropriate action after that.

The authors show that their Neural Network approach to the SSM/I remote sensing problems gives best results in both clear and clouded cases. Especially at high wind speeds, when the physics and therefor the models based on physical considerations become much more complicated, the Neural Network model performs much better than any other model.

This example does not show a particularly intricate Neural Network, but it accentuates the diversity in capabilities of simple Neural Networks. This makes it an even more promising tool, as the generic version can with little modifications be used in real life situations.

4 Discussion

We have seen that Neural Networks can be applied in Ocean Modelling in a wide range of research questions. The level of maturity of the Neural Network method varies, from very unmature in the wind wave modelling to fully developed in the UNESCO equation of state problem.

The Neural Network technique looks promising for the three examples given here, and each have their own peculiarities and problems, but there are a few general issues that can be addressed.

It is not yet fully understood how the architecture of the Neural Networks (number of hidden neurons) influences the outcomings. There are also no general rules on what architecture to pick for a specific problem.

Training a Neural Network is not an easy job. Selecting a correct and representative training set requires good understanding of the variables that are important in the research. It has no use to include variables in the training set that are not even remotely important in the process to be modelled. On the other hand, one should be careful not to exclude potentially important parameters.

The length of the training is important, since a too short time will make the training algorithm not to converge and a too large time can overfit the data. Choosing the right length is still a manner of trial-and-error.

The Neural Network is often still a 'black box'. It maps the input vector on the output vector, but the physics behind it will remain a mystery. This is both an advantage, since it requires no *a priori* choices, and a disadvantage. The purpose of research should always be about understanding the physics behind a process, not to construct some handy tool. There are people investigating how to extract physics from the biases and weights of the Neural Network, but this is still very immature.

The diversity in the three examples given here gives stimulation to be aware of other problems in ocean modelling where the Neural Network technique can be applied. Perhaps a Neural Network point-of-view on some difficult problems facing the community can give new insights as on how to solve them. An ocean modeler should therefor always have the Neural Network technique in his or her mental toolbox.

References

M.W. Gardner and S.R. Dorling. Artificial neural networks (the multilayer perceptron) – a review of applications in the atmospheric sciences networks (the multilayer perceptron) – a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32, 1998.

A.E. Gill. *Atmosphere-Ocean Dynamics*. International Geophysics Series. Academic Press, 1982.

M.S. Griffies, C. Böning, F.O. Bryan, E.P. Chassignet, R. Gerdes, H. Hasumi, A. Hirst, A-M. Treguier, and D. Webb. Developments in ocean climate modelling. *Ocean Modelling*, 2, 2000.

S. Hasselman, K. Hasselman, J.H. Allender, and T.P. Barnett. Computations and parameterizations of the nonlinear energy transfer in a gravity-wave spectrum, part ii: parameterizations of the nonlinear energy transfer for application in wave models. *Journal of Physical Oceanography*, 15, 1985.

W.W. Hsieh. Nonlinear principal component analysis by neural networks. *Tellus, Ser. A.*, 53, 2001.

W.W. Hsieh. Nonlinear multivariate and time series analysis by neural network methods. *Review of Geophysics*, 42, 2004.

W.W. Hsieh and B. Tang. Applying neural network models to prediction and data analysis in meteorology and oceanography. *Bulletin of the American Meteorological Society*, 7, 1998.

V.M. Krasnopolsky, D.V. Chalikov, and H.L. Tolman. A neural network technique to improve computational efficiency of numerical oceanic models. *Ocean Modelling*, 4, 2002.

V.M. Krasnopolsky and F. Chevallier. Some neural network applications in environmental sciences. part ii: advancing computational efficiency of environmental numerical models. *Neural Networks*, 16, 2003.

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

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning internal representations by error propagation*. MIT Press, 1986.

H.L. Tolman, V.M. Krasnopolsky, and D.V. Chalikov. Neural network approximations for nonlinear interactions in wind wave spectra: direct mapping for wind seas in deep water. *Ocean Modelling*, 8, 2005.