10 Implementing a Neural Network Emulation of a Satellite Retrieval Algorithm

George S. Young

10.1 Introduction

As shown in Stogryn et al. (1994), Krasnopolsky et al. (1995), Thiria et al. (1993), Cornford et al. (2001), and many other studies summarized in Chapter 9, neural networks (NN) can be used to emulate the physically-based retrieval algorithms traditionally used to estimate geophysical parameters from satellite measurements. The tradeoff involved is a minor sacrifice in accuracy for a major gain in speed, an important factor in operational data analysis. This chapter will cover the design and development of such networks, illustrating the process by means of an extended example. The focus will be on the practical issues of network design and troubleshooting. Two topics in particular are of concern to the NN developer: computational complexity and performance shortfalls. This chapter will explore how to determine the computational complexity required for solving a particular problem, how to determine if the network design being validated supports that degree of complexity, and how to catch and correct problems in the network design and developmental data set.

As discussed in Chapter 9, geophysical remote sensing satellites measure either radiances using passive radiometers or backscatter using a transmitter/receiver pair. The challenge is then to estimate the geophysical parameters of interest from these measured quantities. The physics-based forward problem (equation 9.4) captures the cause and effect relationship between the geophysical parameters and the satellite-measured quantities. Thus, the forward problem must be a single-valued function (i.e. have a single possible output value for each set of input values) if we have access to all of its input parameters. As a result, the forward problem is generally well-posed, i.e. variations in the input parameters are not grossly amplified in the output. One could, however, imagine some geophysical processes for which the forward problem was ill-posed for some parameter values as a result of a sudden transition from one regime of behavior to another (e.g. the onset of fog formation producing a sharp change in shortwave albedo in response to a minor change in air temperature).

In contrast to the forward problem, the inverse problem of deducing geophysical parameters from satellite measurements does not follow cause and effect and so is not necessarily single-valued over the bulk of the input parameter space. Thus, these satellite retrieval problems are often ill-posed over a significant part of their input parameter space. As a result, the transfer function (equation 9.5) can be multi-valued, having a finite number of solutions at each point in some regions ofthe input parameter space and an infinite number of solutions at points along the borders of these regions. These issues must be addressed via some form of regularization (i.e. using prior knowledge of the problem's physics to constrain the solution in some way). The extended example described below exhibits all of these problems.

The example examined here is the classic problem of retrieving sea surface wind speed from satellite measurements of sea surface backscatter. In this case, a Synthetic Aperture Radar (SAR) is used to measure the backscatter from short-wavelength, wind-driven

George S. Young (✉)
Department of Meteorology, The Pennsylvania State University, PA, USA
Address: 503 Walker Building, University Park, PA 16802, USA
Phone: 814-863-4228; fax (814) 865-9429;
email: young@meteo.psu.edu

ocean waves. Because the wave amplitude is driven by the wind-induced surface stress, the forward or causal direction is from wind speed to backscatter. Thus, the problem of interest for remote sensing is the inverse or retrieval problem.

The traditional approach to solving the SAR wind speed retrieval problem makes use of a semi-empirical model that exists for the forward problem (Monaldo et al. 2001). The version used here is called CMOD4 (Stoffelen and Anderson 1997a, b). This forward model is, however, too complex to invert analytically, so that approach can't be used to produce an analytic model for the inverse problem. Instead, the traditional approach is to run the forward problem for the full range of expected wind speeds, and then select the speed for which it predicts a backscatter value closest to that observed. This approach involves running the computationally expensive physics-based forward model many (typically 25) times for each pixel in a SAR image, thus restricting realtime applications and bulk data analysis.

NN, however, are well suited to emulating smooth nonlinear functions such as that in the forward (CMOD4) model (see Chapter 9). This raises a number of questions concerning their potential for improving the efficiency of the wind speed retrieval process. Could we achieve speed up (i.e. use fewer or faster floating point operations and intrinsic function calls) by using a NN to emulate the forward model? Could we go one step further and use a NN to emulate the inverse model even though its analytic form is not known? Because we have a physics-based analytic model for the forward problem we can generate the data required to train a NN for either the forward or inverse problem as discussed in Chapter 9.

There are, thus, several ways in which NN could be applied to solve the SAR wind speed retrieval problem. First, one can emulate the forward model with a NN and retain the current brute-force-search approach (i.e. examining each of the 25 candidates) to finding a wind speed that yields the observed backscatter. While likely to succeed because the forward physics is generally well-posed, this approach maintains the fundamental inefficiency of the traditional method, using many forward model runs to obtain the answer for each pixel. Second, one can emulate the inverse model with a NN. This is a potentially more challenging problem because the physics is apt to be ill-posed in at least part of the input parameter space. It does, however, offer a distinct advantage in terms of efficiency because only one model run is required per pixel. We'll explore the pros and cons of these two approaches as the example unfolds below. Third, one can follow the approach suggested by Thiria et al. (1993) for similar problems in scatterometry, emulating the inverse model with a categorization rather than regression NN. This approach only partially mitigates the inefficiency of using a multi-run forward model to solve the retrieval problem because, although the NN is run only once, the network's output layer has a node for each of the discretized values of wind speed. The activation output of each of these output nodes is the coefficient of likelihood for the corresponding value of wind speed. A continuous value of wind speed is obtained by fitting an interpolation curve to the peak of this likelihood function. The Thiria et al. method does, however, offer the interesting benefit of documenting whether the inverse function is single- or multi-valued: a single peak will occur in the plot of likelihood coefficient versus wind speed if inverse function is single-valued and multiple peaks will occur if it is multi-valued. Thus, this third approach can serve a valuable diagnostic role, guiding the development of more efficient methods based on the inverse regression model approach.

10.2 Method

The success of a NN development project depends crucially on two factors, use of an appropriate training data set and design of a minimally complex, but sufficient network. As we shall see as the example progresses, both of these factors can pose serious challenges in ill-posed retrieval problems.

10.2.1 Developmental Data

As pointed out in Chapter 9, the existence of an analytic forward model such as CMOD4 offers several advantages when developing a NN for the corresponding retrieval problem. Using the forward model to generate the training datasets allows us to cover the entire parameter space with cases, including types of events that would be rare in observations. This eliminates one of the common problems encountered when training a NN from observations wherein the observation set, and thus the network's fitness, is dominated by the climatologically most likely parts of the input parameter space. By using the physics-based forward model to uniformly cover all parts of the parameter space with cases, we can ensure that the NN is trained to perform as well in the rarer parts of the input parameter space as it does in the more commonly encountered cases. A NN trained in this manner functions in interpolation mode for all cases rather than having to extrapolate from the training set when dealing with the rarer events. As discussed in Chapter 9, this approach leads to better generalization, i.e. better performance in actual operations.

For the SAR wind speed problem the parameters are surface wind speed, backscatter, incidence angle, and the radar look direction relative to the surface wind direction (Monaldo et al. 2001). For the forward model all but backscatter are input parameters and for the inverse model all but wind speed are. Thus, to obtain developmental data for both the forward and inverse models, CMOD4 was used to create a uniform four-dimensional grid of backscatter values with cases spaced every 1° in incidence angle, every 1° in radar look angle relative to the wind direction, and at 450 values of wind speed from 0 to 60 ms^{-1}. This results in 5,670,000 cases, more than can be processed in a reasonable time via the NN training algorithm. Therefore two samples of 10,000 cases are selected at random, one for training the network, a second for validation, and a third for final validating. Given the independence of the samples, high performance on the validation set indicates that the NN has been fit robustly across the input parameter space. Of course, if the same validation set is used repeatedly when optimizing the NN architecture, the resulting NN will be biased towards it as it is towards the developmental data. Therefore, a third set of 10,000 cases is used to recalculate the skill of the final network, eliminating this source of false optimism. Alternatively, one could use a different validation set on each of the NN architectures in order to achieve this independence.

10.2.2 Neural Network Design

The basics of NN are discussed in chapter Y-NN and one approach to universal function emulation using NNs is explored in Chapter 9. Although details can vary from one implementation to another, the basics are the same. The network's input layer consists of a data source for each of the input parameters. All of these values are fed into each of the processing nodes of the next layer, often called a hidden layer. Each processing node first computes the output of a linear equation, which takes its input from the input layer. The coefficients can differ between processing nodes, so differing computations are done in parallel by the nodes of a processing layer. The results of each equation are then "squashed" into the range -1 to 1 using the hyperbolic tangent or a similar activation function. At this point the squashed results may be fed into an output layer as in Chapter 9 or into additional processing layers as in Reed and Marks (1999). The latter approach will be used in the example discussed in this chapter. In either case, the results of the final processing layer are fed through one more linear equation, called an output node. If the result is to be a probabilistic or categorical (yes/no) prediction, than a squashing function is applied to the output layer's result. If, as in the SAR wind speed retrieval problem, the desired output is numerical, no squashing function is applied to the output node.

Training of the NN involves finding values for all of these coefficients. This can be done via various forms of the back propagation of errors (Reed and Marks 1999) or by using a general purpose nonlinear optimization tool such a genetic algorithm (GA, Haupt and Haupt 2004) (Jones 1993; Munro 1993). Neither of these approaches is necessarily best for all retrieval problems. For example, training a NN to emulate some nonlinear functions is "GA-easy" while training the network to emulate other functions of equivalent complexity is "GA-hard". The "GA-hard" problems are those where the under-constrained nature of NNs leads to multiple distinct solutions, each at a local minimum in the network's forecast error, that, when bred, yield solutions of lower rather than higher skill (Reed and Marks 1999). Likewise networks trained by back propagation can become trapped in such local minima (Yamada and Yabuta 1993; Reed and Marks 1999). Back propagation techniques are popular, particularly in the data mining community, (e.g. Alpsan et al. 1995; Reed and Marks 1999; Witten and Frank 2005) and will be employed here.

As mentioned in Chapter 9, NNs with a single hidden layer are adequate to fit any continuous nonlinear function provided enough hidden nodes are used.

Thus, the design process for a single-layer network consists of simply increasing the number of nodes until good performance is achieved on independent validation data.[1] The smallest network that achieves good performance is then used in practice, both because it is more efficient than larger networks and because it is less likely to be over-fit and so will generalize better. The design problem is more complex when the network includes multiple processing layers. The designer not only has to decide on the number of layers, but also on the number of nodes per layer. The design space is thus multidimensional. One could, of course, use a genetic algorithm or some other general purpose nonlinear optimizer to develop the network design (Dodd 1990; Reed and Marks 1999), but in practice one often simply tries a number of likely candidates learning enough from each to design a likely successor.

While this iterative design process includes two relatively distinct aspects, determining how many layers to use and determining how many nodes to include in each, there is a tradeoff between the two because increasing the number of layers decreases the number of nodes required in each. The design tactic employed here is to alternately tune the number of nodes and the number of layers, starting first with the nodes and then cycling back and forth until performance stops improving. Experience suggests that three hidden layers makes a good starting point with the first layer having as many nodes as there are predictors and the other layers progressively less. The network designer then tunes the number of nodes per layer by trying greater and lower node counts in the networks and seeing which produces the best result. This is just the manual version of the various gradient descent methods common to nonlinear optimization (Nelder and Mead 1965). Once the optimal number of nodes is found, one should check the weights in the last hidden layer. If each node has the same weight values as all the others, then they're all making the same calculation and the layer is probably redundant. In some cases the results will improve by eliminating that layer and in other cases by reducing the number of nodes it contains until the weights differ from node to node. If the nodes in the last hidden layer differ, one could try adding an additional layer of two nodes. If performance improves, one could start tuning the number of nodes again, otherwise go back one design and stop iterating. If this approach seems too complex, stick with a single hidden layer as described in Chapter 9.

10.2.3 Network Training

We could easily develop our own back propagation training program following, for example, Reed and Marks (1999). Ready built open source tools such as Weka (Witten and Frank 2005) are, however, readily available for download (http://www.cs.waikato.ac.nz/ml/weka/), so that approach is often more efficient. Weka will be used here because it offers good documentation, a point-and-click interface, and extensive support for troubleshooting and validation. No matter which development tool we use, the phases are the same.

- Data ingest
- Predictand and predictor selection
- Cross-validation setup (in this example using a validation set)
- Network design
- Network building (i.e. tuning the regression coefficients)
- Network testing on independent validation data

Following each validation stage the network is redesigned as described above and the build and validate cycle is repeated. Clearly, to make this cycle converge, careful consideration must be given to how the validation results vary with network design. The final design must yield a high accuracy solution but do so with the minimum number of nodes, thereby minimizing computational cost.

10.3 Results

The NN development tactics described above are applied to both the forward and inverse problems linking SAR backscatter with surface wind speed. The results will be presented in the order originally encountered when this project was under development. Each of the problems that arose during this development

[1] In a more general setting, for example with noisy data, the optimal NN is the one that has the lowest average error over multiple validation sets.

process will be discussed at the stage where it was first diagnosed. Thus, the results discussion will focus on how the problems were detected and then addressed by appropriate regularization. The section will conclude with a discussion of efficiency issues and a comparison of the various NN approaches to the traditional CMOD4 approach they are designed to replace.

10.3.1 Forward Model

Table 10.1 shows a sample of network designs for the forward (wind speed to backscatter) problem. The network design is described via a comma separated list giving the number of nodes in each hidden layer. Training was undertaken for 500 cycles (Epochs) in all but one case. The model performance is described in terms of percent variance explained (i.e. r^2) and mean absolute error (MAE). Each network's computational cost is computed from a count of the various floating point operations and intrinsic function calls using times estimated for each of these actions via a procedure described in Section 10.3.4 below. The cost value is then normalized by the similarly computed run time estimate for the analytic CMOD4 code. Lower values of cost are of course to be preferred over high values. Note that these values are not the cost of network development, but rather the computational cost to apply the network in operational SAR image analysis.

The first NN, with a single hidden layer of four nodes shows skill, but is not accurate enough to serve as a replacement for the analytic CMOD4 forward model as the MAE remains large relative to operational requirements. Performance improves when two more hidden layers are added and further improves when the number of nodes in these layers is increased past a total of 20. There is, however, some decrease in skill for the most complex networks, suggesting either over-fitting or an inadequate number of training epochs for a network of that degree of network complexity. Because the skill improves markedly when the most complex network is redeveloped using 5,000 epochs instead of 500, we can conclude that inadequate training rather than over-fitting was the problem. This final NN was reevaluated on an independent set of test data yielding a percent variance explained of 0.9998, the same value obtained with the validation data set. Thus, we can conclude that the NN was not over-fit, a fact attributable to the noise free synthetic data and the large number of cases in the developmental data set. The MAE of 0.0052 is accurate enough that the NN could be used as a replacement for the analytic CMOD4 forward model. It is, however, a factor of five more costly in computational time than the analytic CMOD4 model. Clearly, if NNs are to be of aid in the SAR wind speed analysis it must be in the inverse rather than forward problem, where it would replace multiple runs of the analytic CMOD4 model instead of just one.

10.3.2 Inverse Model

Given the successful emulation of the analytic CMOD4 forward model, it is reasonable to ask whether the neural net approach can be applied to emulate the inverse problem for which no analytic model exists. Doing so would eliminate the need for multiple runs of the analytic forward model to make up for the lack of an analytic inverse model. As Table 10.2 shows, however, the initial NN results are quite disheartening.

The skill demonstrated by each NN was markedly worse on the inverse problem than it had been on the forward problem discussed in the section above.

Table 10.1 Neural net design and performance for the forward problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	r^2	MAE	Relative cost
4	500	0.9577	0.15	0.28
4,3,2	500	0.9616	0.10	0.63
8,4,2	500	0.9631	0.12	1.03
12,6,2	500	0.9956	0.050	1.55
16,8,4	500	0.9972	0.023	2.29
24,12,2	500	0.9984	0.015	3.98
32,16,2	500	0.9962	0.035	4.84
32,16,4	500	0.9884	0.043	5.06
32,16,4	5,000	0.9998 (0.9998)	0.0053 (0.0052)	5.06

Table 10.2 Neural net design and performance for the inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	r^2	MAE (ms^{-1})	Relative cost
12,6,3	500	0.8188	5.02	0.0912
16,8,4	500	0.8714	4.03	0.1281
24,16,2	500	0.8712	3.75	0.2195
32,16,2	500	0.8571	4.53	0.2713
32,16,4	500	0.8709	4.12	0.2832
32,16,4	5,000	0.8541 (0.8506)	3.80 (3.82)	0.2832

Indeed, the best r^2 value was near 0.87 rather than above 0.999. Another sign that the NN is not performing well is that further training on the inverse problem did not provide a performance improvement the way it did on the forward problem. Thus, something about the inverse problem is making it much more challenging for neural net emulation. The problem cannot be attributed to reuse of the validation data because similar skill was obtained for the most complex network when it was reevaluated using an independent set of test data.

Examination of the relationship between wind speed and backscatter in CMOD4 quickly reveals the source of the problem. For a number of incidence angle and wind direction combinations, such as that shown in Fig. 10.1, the relationship is multi-valued in wind speed. Thus, while each wind speed corresponds to one backscatter value, a single backscatter may correspond to two widely different wind speeds. Because the NN has only one output it can't fit a multi-valued function, and thus fails over much of its input parameter space.

There are two obvious approaches for solving this problem. One approach is to redesign the training set and the NNs to support two output values. These values would differ for those parts of the input parameter space where wind speed is a multi-valued function of backscatter, while being the same where the function is single-valued. This approach leaves the decision as to which value is correct to the end user, thus putting off the required regularization of the inverse problem. The second approach is to regularize the data in the training set as described in Chapter 9, so that the NN is developed to return only one of the two possible wind speed values. This approach most closely follows the

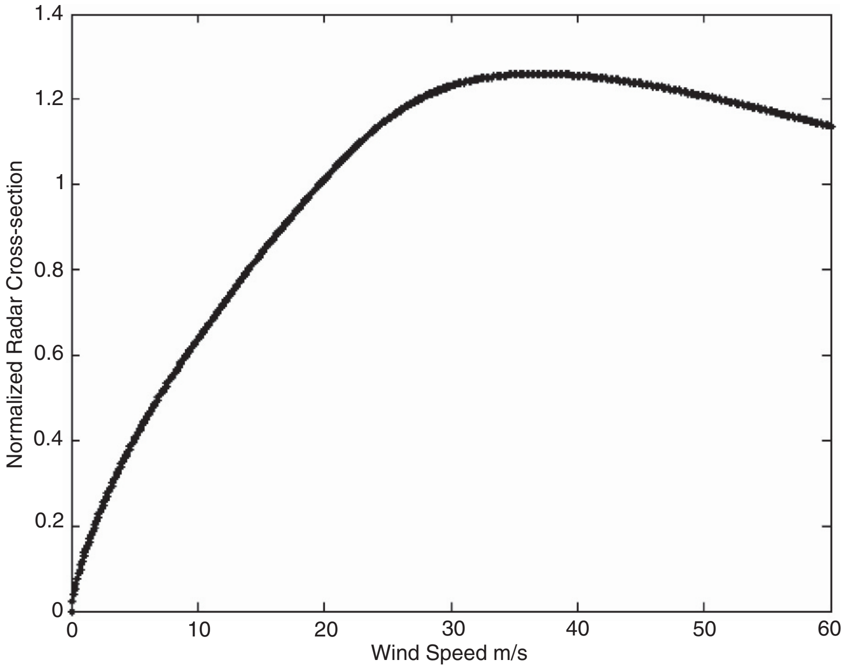

Fig. 10.1 Normalized radar cross-section (i.e. backscatter) as a function of wind speed for an incidence angle of 20° and a wind-relative look direction of 45°

current practice with the analytic CMOD4 model and so will be followed in the sections below.

10.3.3 Regularization of the Inverse Model

A simple regularization of the training data set can be achieved by eliminating those cases for which the wind speed takes on the higher of two possible values. This follows the current CMOD4 practice (N. S. Winstead 2006, personal communication) of assuming that the lowest wind speed consistent with the backscatter is most likely to be correct. This regularization makes the function to be emulated single-valued and thus should improve the NN's performance on the inverse problem.

As shown by comparing Tables 10.2 and 10.3, however, this improvement does not occur (except for the r^2 statistic for one network design). Rather, the percent variance explained dropped slightly and the MAE increased. Why did the regularization fail to make the problem more tractable? Reexamination of Fig. 10.1 sheds light on the problem. Not only is wind speed potentially a multi-valued function of backscatter, but also it exhibits a zone of hypersensitivity to backscatter near the peak of this latter parameter. Thus, for backscatter observations near this peak value, major changes in wind speed result from minor changes in backscatter. So backscatter provides little information on wind speed in that zone. This sensitivity makes the NN hard to train and would in any case cause the resulting network's output to be highly sensitive to observational errors in its input. This sensitivity rule applies not just to NNs, but to any model as mentioned in the discussion of ill-posed systems in Chapter 9.

The lesson to be learned from this failure is that if the gradient of input with respect to output is small, then output is hypersensitive to input because the gradient of output to input is huge. Thus, careful examination of the Jacobian of the problem can greatly aid in planning a successful regularization strategy. This task can be undertaken either numerically, or by examination of the plotted function as was done above.

Full regularization requires elimination of these zones of hypersensitivity as well as resolution of any regions where the function to be emulated is actually multi-valued. Users of the analytic CMOD4 model do this by limiting analysis to wind speeds of less than some threshold, typically 25 ms^{-1} (N. S. Winstead 2006, personal communication). It is simple enough to apply this same approach to neural net development, eliminating all cases with higher wind speeds from the training. Note that there are many other forms of regularization. This is just the one that's appropriate for the SAR inverse problem. Note also that, as pointed out in Chapter 9, the resulting NN is apt to be wildly inaccurate outside of its training domain. Thus, in making operational use of the NN, all output wind speeds above 25 ms^{-1} should be truncated to that value and the value itself viewed as an out-of-domain error flag. This is the current practice with the analytic CMOD4 model, so the NN results shown in Table 10.4 are a fair test of performance relative to the analytic code.

This time the regularization achieves its desired effect. The skill of the fully regularized inverse model is almost as good as that of the forward model. This result holds up when the network is reevaluation on an independent set of test data, demonstrating that over-fitting did not take place. Some hint of the challenge remains in that it took an order of magnitude more training epochs to achieve near perfection on the percent variance explained statistic. The best network yielded an MAE of about 0.1 ms^{-1}, about one tenth of the typical error of the CMOD4 analytic model in comparison with observations (Monaldo et al. 2001).

Table 10.3 Neural net design and performance for the single-valued inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	$\mathbf{r^2}$	MAE (ms^{-1})	Relative cost
12,6,3	500	0.8521	8.98	0.0913
16,8,4	500	0.8545	7.40	0.1281
24,16,2	500	0.8321	5.05	0.2195
32,16,2	500	0.8697	7.75	0.2713
32,16,4	500	0.8722	7.32	0.2832
32,16,4	5,000	0.8668 (0.8673)	5.37 (5.33)	0.2832

Table 10.4 Neural net design and performance for the fully regularized inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	r^2	MAE (ms^{-1})	Relative cost
4,2	5,000	0.8416	2.67	0.0235
12,6,3	5,000	0.9857	0.77	0.0913
16,8,4	5,000	0.9878	0.57	0.1281
20,10,5	5,000	0.9894	0.56	0.1681
20,20,2	5,000	0.9984	0.42	0.2208
24,16,2	5,000	0.9988	0.34	0.2195
27,9,3	5,000	0.9974	0.24	0.1886
32,16,2	5,000	0.9982	0.19	0.2713
32,16,2	50,000	0.9990 (0.9990)	0.10 (0.12)	0.2713

Thus, emulation of the CMOD4 analytic model by an appropriate NN would not substantially increase error in operational applications. The remaining question is then, would such a replacement be more efficient in terms of computational cost.

10.3.4 Computational Cost

Estimation of wind speed from SAR backscatter using the analytic CMOD4 forward model is a costly process, both because the model must be run for each candidate wind speed and because the model itself includes a number of calls to costly intrinsic functions such as cosine and hyperbolic tangent. The first problem can be solved by using a NN to emulate the inverse model as discussed above. Unfortunately, from the computational cost perspective, each NN node includes a call to the hyperbolic tangent intrinsic function and so is, itself, fairly costly. An estimate of the computational cost of the existing CMOD4 code and the NN inverse model can be obtained by counting the number of these operations in each approach.

The computational cost for each of the floating point operations and intrinsic function calls is obtained by a benchmarking program that executes each instruction enough times to obtain accurate timing information. The values shown in Table 10.5 were obtained using the default floating point variables in MATLAB version 7 running on a 3.2 GHz Intel Pentium 4 Windows XP PC. These timing results will of course vary with computer language, hardware, and floating point precision, but the final results reported in Tables 10.1 through 10.4 are, to a certain extent, normalized when the NN time estimates are scaled by that for the analytic CMOD4 model. The full operation count for CMOD4 and a multi-layer NN are also shown in Table 10.5.

The key result of this timing analysis is that intrinsic function calls such as cosine and hyperbolic tangent are much more costly than the basic floating point operations of add, multiply, and divide. Thus, the NN times reported in Tables 10.1 through 10.4 are dominated by the hyperbolic tangent calls and are therefore approximately linear with the number of nodes.

Table 10.5 Inputs for the computational cost calculation for CMOD4 and the neural networks. The functions in this table are those used in the analytic CMOD4 code and the neural network

Operation or intrinsic function call	Time in nano-seconds	Uses in CMOD4	Uses in neural network
Add	7	31	Sum over all layers of number of inputs plus one times number of nodes
Multiply	11	36	Sum over all layers of number of inputs times number of nodes
Divide	13	4	0
Cosine	117	2	0
Tangent	174	2	0
Power	477	4	0
Log to base 10	462	1	0
Square root	144	1	0
Hyperbolic tangent	359	2	Equals number of hidden nodes

A 32,16,4 network has 52 such calls and would thus require roughly twice the computer time as a 16,8,2 network. In comparison, the CMOD4 analytic model has only two hyperbolic tangent calls, but does invoke a number of other expensive intrinsic functions.

The key advantage of the neural net inverse model over the analytic CMOD4 model is that it need be run only once instead of 25 times. Thus, even the most complex networks shown in Tables 10.1 through 10.4 require only one fourth of the time of the analytic model. As a result, replacement of the analytic CMOD4 model by a NN inverse model can yield a substantial savings in computational time at the price of a modest increase in wind speed retrieval error.

In light of NN's success on the problem, it is worth considering whether more traditional methods could have achieved a similar speedup in the SAR wind speed retrieval. For example, the search strategy used in the operational CMOD4-based retrieval is to run the analytic forward model at $1\,\mathrm{ms}^{-1}$ intervals from 1 to $25\,\mathrm{ms}^{-1}$. If the more efficient binary search algorithm (Knuth 1997) was used instead, only seven runs of the forward model would be required to bracket the answer to this precision, not 25. This speed ratio of 0.28 is almost identical to the 0.27 achieved by the high accuracy neural net emulation. Thus, there are multiple means of accelerating the SAR wind speed retrieval process. The sole obvious advantage of a NN over binary search is that it does not require branching code and so should pipeline better on modern microprocessors.

10.4 Conclusions

As demonstrated in Chapter 9 and the example above, NNs can provide accurate emulations of smooth nonlinear functions, as long as they are well-posed. NNs can exhibit a substantial speed advantage when computing the desired function would be otherwise cumbersome, for example when the traditional approach involves iterative improvement or examination of many trial solutions. The latter situation occurs for the example problem described above, so a NN with one fourth of the computational cost can emulate the operational retrieval algorithm to within one tenth of the operational model's own accuracy. So, for a slight increase in error and total loss of physical insight, the NN provides a massive improvement in computational efficiency. In contrast, neural net emulation of the forward problem exhibits a substantial cost penalty. This difference between inverse and forward problems results from the efficiency of the analytic forward model, CMOD4, and the inefficiency of the multi-run brute force search strategy traditionally used in applying it to the inverse problem. Interestingly, replacing the brute force search with a binary search using the analytic forward model would yield a run time improvement almost identical to that achieved via NN emulation. Clearly the paths to wisdom are many and varied.

Acknowledgements This work was supported by NSF and ONR through grants ATM-0240869 and N00014-04-10539. Special thanks are due to Nathaniel Winstead and Frank Monaldo of the Johns Hopkins University Applied Physics Laboratory for providing the CMOD4 analytic code and numerous valuable insights.

References

Alpsan, D., Towsey, M., Ozdamar, O., Tsoi, A., & Ghista, D. (1995). Are modified back-propagation algorithms worth the effort? *IEEE International Conference on Neural Networks, Orlando, USA*, *1*, 567–571.

Cornford, D., Nabney, I. T., & Ramage, G. (2001). Improved neural network scatterometer forward models. *Journal of Geophysical Research 106*, 22331–22338.

Dodd, N. (1990). Optimization of network structure using genetic techniques. *Proceedings of the International Joint Conference on Neural Networks, Washington D.C., USA*, *1*, 965–970.

Haupt, R., & Haupt, S. (2004). *Practical genetic algorithms* (2nd ed., 253 pp.). Hoboken, NJ: Wiley.

Jones, A. (1993). Genetic algorithms and their applications to the design of neural networks. *Neural Computing and Applications*, *1*, 32–45.

Knuth, D. (1997). *The art of computer programming, volume 3: Sorting and searching* (3rd ed., 780 pp.). Reading, MA: Addison-Wesley.

Krasnopolsky, V., Breaker, L., & Gemmill, W. H. (1995). A neural network as a nonlinear transfer function model for retrieving surface wind speeds from the special sensor microwave imager. *Journal of Geophysical Research 100*, 11033–11045.

Monaldo, F., Thompson, D., Beal, R., Pichel, W., & Clemente-Colón, P. (2001). Comparison of SAR-derived wind speed with model predictions and ocean buoy measurements. *IEEE Transactions on Geoscience and Remote Sensing*, *39*, 2587–2600.

Munro, P. (1993). Genetic search for optimal representations in neural networks. In R. Albrecht, C. Reeves, & N. Steele

(Eds.), *Artificial neural nets and genetic algorithms. Proceedings of the international conference* (pp. 628–634). Innsbruck, Austria: Springer.

Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*, 308–313.

Reed, R., & Marks, R. (1999). *Neural smithing: Supervised learning in feedforward artificial neural networks* (346 pp.). Cambridge, MA: MIT Press.

Stoffelen, A., & Anderson, D. (1997a). Scatterometer data interpretation: Measurement space and inversion. *Journal of Atmospheric and Oceanic Technology*, *14*, 1298–1313.

Stoffelen, A., & Anderson, D. (1997b). Scatterometer data interpretation: Estimation and validation of the transfer function CMOD4. *Journal of Geophysical Research*, *102*, 5767–5780.

Stogryn, A. P., Butler, C. T., & Bartolac, T. J. (1994). Ocean surface wind retrievals from special sensor microwave imager data with neural networks. *Journal of Geophysical Research*, *90*, 981–984.

Thiria, S., Mejia, C., Badran, F., & Crepon, M. (1993). A neural network approach for modeling nonlinear transfer functions: Application for wind retrieval from spaceborne scatterometer data. *Journal of Geophysical Research*, *98*, 22827–22841.

Witten, I., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed., 525 pp.). San Francisco: Morgan Kaufmann.

Yamada, T., & Yabuta, T. (1993). Remarks on neural network controller which uses genetic algorithm. *Proceedings of International Joint Conference on Neural Networks* (pp. 2783–2786). Japan: Nagoya.