A comparison of neural network approaches for on-line prediction in IGRT

J.H. Goodband, O.C.L. Haas, J.A. Mills

1. Introduction

Whilst it is desirable to achieve a high level of accuracy when delivering X-rays using intensity-modulated radiation therapy (IMRT) techniques, if exact tumour position relative to an irradiated volume is unknown, theoretical advantage in accuracy compared with less complex techniques may be compromised or even negated. Tumour position varies both between treatments (inter-fractional movement) and during treatments (intra-fractional movement). The movement is caused by a combination of daily changes in patient physiology, e.g. bladder size; continuous changes e.g. breathing; and also by set-up errors associated with treatment complexity. Some patient movement (e.g. head, arm, leg) can be limited through the use of suitable restraints. When tumour movement is caused by breathing, either directly, (cancer of the lung), or indirectly (structures within the thorax and abdomen) the problem becomes more challenging.

To allow for uncertainty in true tumour position, a margin of error, typically between 10 and 15 mm,[1] is added to the perimeter of a clinical target volume, the total volume being classed the planning target volume (PTV). Since the margin represents healthy tissue which ideally should be avoided during treatment, any methods which can be employed to reduce margin size will also benefit the patient. Breath holding has been investigated,[2] but can only be implemented for relatively short treatment sessions and where the patient is healthy enough to comply with the method. Where implementation of breath holding is impractical, accurate real-time knowledge of tumour position is vital if margin size is to be reduced.

Adaptive radiation therapy (ART)[3] and image guided radiation therapy (IGRT)[4] are recent innovations which attempt to reduce errors by imaging the tumour, either immediately prior to (inter-fractional), or during (intra-fractional) treatment. Ultrasound imaging,[5] cone beam CT[6] or CT scans[7] can be used for inter-fractional tumour positioning. On-line intra-fractional organ motion detection can be facilitated using either fluoroscopic X-rays for observing internal markers[8] or stereoscopic video techniques for tracking external markers.[9] Positional information gained on-line can then be used to reduce treatment errors by modifying the beam position and/or cross-section in real-time. Gating[10] is a method whereby the treatment beam is automatically turned off if the tumour moves out of a pre-determined gating region. Significant improvement in dose conformality with gating implemented has been demonstrated using a moving lung phantom with breathing simulated by the Lujan model.[11] In practice, however, a human patient does not conform to a fixed parameter model, the patient having breathing characteristics which are to a greater or lesser extent unpredictable. Assistance to minimize unpredictibility can be given in the form of breath training prior to treatment. There is, however, some evidence[9] that breath training provides no significant improvement to clinical outcome. In real-time, changes in breathing can be detected by markers, but there is an inevitable lag between observation time and system delivery time. This time delay is known as the *latency* of the treatment system. To provide an effective response to these breathing changes, some form of prediction method is therefore required to compensate for the system latency.

Recent investigations[12–16] of different prediction methods for tracking organ movement have demonstrated the benefits gained by using linear filters, Kalman filters and neural networks (NNs) compared with using no prediction method. Only Isaksson *et al*[14] attempt on-line adaptive updating. NN based methods (Kakar *et al*[15] use an adaptive fuzzy neural inference system, which is an augmented Gaussian radial basis function NN) produce the best root-mean-squared error (RMSE) solutions for latencies > 0.2 s. Whilst providing useful comparisons to be made with alternative methods, RMSE values do not, however, give any indication of the *maximum* error values produced by

the methods used and can therefore potentially lead to misleading conclusions with regard to the local accuracy of a particular training algorithm. A large prediction error even if made for a relatively short period of time may cause the treatment beam to deliver a high dose to an organ at risk (OAR) with possibly harmful short or long-term effects. It should also be noted that although a well trained NN may be accurate for *interpolation* purposes, the same NN can potentially be extremely inaccurate when used for *extrapolation*. To avoid extrapolation errors a NN used for prediction in a non-stationary dynamic system should be updated regularly to account for changes in signal characteristics.
The overall aim of this paper is to highlight potential issues with NN methods which have previously been employed to predict organ motion and to propose a more rigorous approach when using NNs in RT. To this end, the paper commences with a brief review of NN theory relevant to time series prediction which emphasizes the importance of choosing an appropriate algorithm for adaptive training. Subsequently, the performance of a time-series prediction neural network (TSP NN)[17] for estimating tumour marker position is assessed using three different methods to up-date parameters on-line. Of these, the *generalized regression* NN has not previously been investigated for use in IGRT. The training data used have been previously published by George *et al*[18] and are taken as a benchmark to demonstrate the relative benefits of the different approaches when applied to the same data. In contrast with previous work, relative accuracy is assessed both in terms of root-mean-squared and maximum errors.

2. Methods and materials

This section presents a short introduction to NNs followed by details of the training data, NN design and algorithms used in the present study.

1. Neural networks

The neural network (NN) is a method by which a computer can be trained to 'learn' a relationship between input and output data using mathematical activation functions and coefficients (generally called *weights*) designed to simulate the interactions between neurons and synapses in the human brain. This makes the NN a very powerful tool in situations where an explicit functional relationship between inputs and outputs is non-linear and difficult to determine analytically. If sufficient input/output data is readily available, the training of a NN may provide the best combination of speed and accuracy to replicate a desired function without the requirement for intrinsic knowledge of the function from the operator. This does not mean, however, that NNs should be implemented without knowledge of their limitations. The present work aims to provide sufficient information to allow an unbiased appraisal of their abilities within the specific area of tumour motion prediction.

1. Network architectures

A NN is a combination of inputs, weights, activation functions (generally called neurons) and outputs. The way in which these attributes are combined defines the *architecture* of the NN. The most popular NN method for synthesizing input-output mappings is *supervised learning* using training data to construct a *perceptron*. The perceptron uses stochastic iterative training algorithms to produce a *global* solution to functions i.e. every input gives a unique output. It should be noted, however, that attempting to predict values using inputs significantly different from the original training data (extrapolation) will usually result in spurious outputs which may be several orders of magnitude different from the values desired. A perceptron consisting only of an input layer and a computational output layer of neurons is called a *single-layer perceptron* i.e. the input layer (which distributes the inputs without performing computation) is not counted when describing the number of layers. In general, multiple inputs may be linked to multiple outputs using more than one layer of neurons – any layer between the input and output layer is called a *hidden* layer. A perceptron with at least one hidden layer is called a *multi-layer* perceptron (MLP).
Conversely, the *radial basis function neural network* (RBF NN) always has one hidden layer of radial basis function neurons and uses deterministic linear algebraic techniques to calculate network weights (although these are sometimes combined with stochastic methods of data selection or clustering). The RBF NN produces a *local* solution to functions i.e. outputs are bounded to be within the range of the training data. This property makes RBF NNs preferable for safety critical prediction.[19]
In a perceptron architecture, the activation function, f, represented by each neuron, is generally a

non-linear *squashing* function

$$f(v) = (1+e^{-v})^{-1},\ f \in (0,1) \forall v \in (-\infty,\infty) \text{or} \tag{1}$$

$$f(v) = (e^{v} - e^{-v})(e^{v} + e^{-v})^{-1},\ f \in (-1,1) \forall v \in (-\infty,\infty) \tag{2}$$

where $v = \sum_{i=1}^{M} w_i x_i = \mathbf{w}^T \mathbf{x}$, i.e. the linear combination of layer weights and M network inputs. (1) and (2) are known as the *logsig* and *tansig* functions respectively. For a RBF NN, the most commonly used function is the Gaussian

$$f(v) = e^{-v},\ f \in (0,1] \forall v \in [0,\infty) \tag{3}$$

where $v = \| \mathbf{c} - \mathbf{x} \|_2^2 / \sigma^2$, **c** being the *centre* of the RBF and σ the *width* of the RBF, representing its degree of response to input vectors and influence on neighbouring layer neurons.

2. Training data

When NNs are required to model an input-output relationship, a set of training data is used which should ideally encode the mapping between all possible inputs and outputs required by the application. This method is known as supervised learning. Data is presented to a NN in the form of a set of inputs and targets (i.e. desired outputs) and a training algorithm is implemented which aims to reduce the value of a scalar energy function, F, generally the mean-squared error (MSE) given by

$$F = \frac{1}{N} \sum_{i=1}^{N} (t_i - y_i)^2 \tag{4}$$

where t_i is a target, y_i is the network response to the i^{th} input, and N is the number of input-target data pairs. This reduction is generally achieved by iteratively changing the values of weights and biases within the NN until either the desired value for F is attained or the algorithm is terminated by an alternative criterion, such as time restriction. Each iteration is called an *epoch*.

The data used for this study have been acquired from a breathing training database collated by George (RGeorge@mcvh-vcu.edu) at Virginia Commonwealth University. 24 adult patients suffering from lung cancer were observed over a period of a year, a record being made of 331 4-minute breathing traces of respiratory motion using 3 types of training. None of the patients were oxygen dependant and all were capable of lying in a supine position without feeling pain. A marker block resting on the chest of each patient between umbilicus and xyphoid allowed tracking of respiratory movement in the anterior-posterior direction using a real-time position management system developed by Varian Medical Systems. The marker position was sampled at 30 Hz. Each record therefore contains $30 \times 4 \times 60 = 7200$ data points. Breathing amplitude (maximum to minimum displacement in each 4-minute trace) ranges from 8 mm to 60 mm. Isaksson *et al*[14] normalize their data (taken from 3 patient records classified as 'easy', 'medium' and 'hard-to-track') to zero mean and unit variance to facilitate more stable NN learning. The method consequently uses knowledge of *future* patient breathing behaviour which is clearly unavailable in a real-time situation. In the present work simulations are carried out in real-time making no assumption of future breathing patterns. The data are not, therefore rescaled, and each NN is required to adapt to any change in breathing characteristic in the absence of *a priori* knowledge. The data are similarly not *a priori* sub-grouped into 'easy' or 'difficult' categories, on the basis that it is only *after* the treatment has been carried out that the degree of tracking difficulty can be realistically assessed. It should be noted that all patients 'behave' (i.e. maintain constant amplitude and frequency) reasonably well in the first 60 s of each record.

2. Time-series prediction neural network

A time series prediction, $\hat{x}(t+T), T$ seconds ahead of the present time t is given by

$$\hat{x}(t+T) = G\{x(t), x(t-\tau), x(t-2\tau), \ldots,$$

$$x(t-(H-1)\tau), x(t-H\tau) + \sum_{h=0}^{H} \varepsilon(t-h\tau)\}$$

$$= G\{\mathbf{x}^T + \sum_{h=0}^{H} \varepsilon(t - h\tau)\} \tag{5}$$

where G is the non-linear function described by the NN, τ is the sampling period, H is the number of historical samples used in combination with the present sample $x(t)$, ε is observation noise, usually assumed to be Gaussian and $\mathbf{x} = [x(t), x(t-\tau), x(t-2\tau), \ldots, x(t-(H-1)\tau), x(t-H\tau)]^T$.

The 'black-box' ability of an NN to model input-output mappings with little or no requirement for *a priori* knowledge has made NNs popular for both modelling and predicting time series data. Consequently, they are frequently used in predictive control.[17,20,21] Each of the following two sections describes a NN design which has been demonstrated as suitable for time series prediction.

(420,200) (34,48) $x(t-H\tau)$ (40,100) $x(t-\tau)$ (50,150) $x(t)$ (90,54)(1,1)95 (90,152)(1,0)94 (90,102)(2,1)95 (90,152)(1,-1)95 (90,54)(1,0)94 (90,102)(2,-1)95 (90,152)(2,-1)94 (90,102)(1,0)94 (90,54)(2,1)95 (125,60) $w^1_{M,H+1}$ (130,108) $w^1_{2,2}$ (130,158) $w^1_{1,1}$ (63,84)*2 (63,78)*2 (63,72)*2 (197,84)*2 (197,78)*2 (197,72)*2 (197,54)24 (197,102)24 (197,152)24 (187,52) f_M (187,100) f_2 (187,150) f_1 (209,102)(1,0)91 (209,152)(2,-1)92 (209,54)(2,1)92 (250,65) $w^2_{1,M}$ (250,108) $w^2_{1,2}$ (250,138) $w^2_{1,1}$ (312,102)24 (306,97) Σ (324,102)(1,0)70 (397,100) y

Figure 1. Time series prediction MLP with one hidden layer. The MLP illustrated represents a 1-D (scalar) predictor with $H+1$ inputs, 1 output, and a hidden layer with M neurons. f represents a non-linear function, generally the logsig or tansig function (equations (1) and (2) respectively).

1. Time series prediction MLP

The MLP architecture to predict time series data constitutes a tapped delay line input, hidden layer the size of which should ideally be optimized using training data, and 1 output (Figure 1). This type of MLP is known as a times-series prediction (TSP) MLP.[17] Although the output need not be scalar, it has been shown that more precise results are obtained using individual networks for each prediction output i.e. MLPs with scalar outputs.[17,19] A TSP MLP is trained by using a sequence of vectors $\{\mathbf{x}_n\}, \mathbf{x}_n \in R^{H+1}, n = 1, 2, \ldots, N$, as inputs, and $\{t_n\} \equiv \{x_n(t+T)\}$ as targets. The accuracy of a TSP MLP is dependent on a number of factors:

1. Signal characteristics
2. Desired prediction horizon
3. Sampling frequency
4. Frequency of on-line updating (if any)
5. Number of inputs $\equiv$ number of historical points used
6. Number of hidden neurons (if any) in network
7. Type of training algorithm implemented

In general, 1-3 are beyond the control of the NN designer. It should be noted that for periodic signals sampling frequency should be at least twice the bandwidth of the signal to avoid aliasing.[22] On-line updating needs to be carried out frequently if the signal is non-stationary, but this frequency may be restricted by the speed of the training algorithm used. Finding optimal solutions for 5-7 can be very time consuming. Any optimization of architecture and training algorithm is generally, therefore, carried out off-line, with on-line up-dating (where carried out) limited to weight changes facilitated by short periods of re-training. As a result, it may be difficult or even impossible for the NN to give an accurate response at points where there is a major change in the signal properties.

(380,200) (-13,48) $x(t-H\tau)$ (-6,100) $x(t-\tau)$ (6,150) $x(t)$ (40,54)(1,1)95 (40,152)(1,0)94 (40,102)(2,1)95 (40,152)(1,-1)95 (40,54)(1,0)94 (40,102)(2,-1)95 (40,152)(2,-1)94 (40,102)(1,0)94 (40,54)(2,1)95 (23,84)*2 (23,78)*2 (23,72)*2 (147,84)*2 (147,78)*2 (147,72)*2 (147,54)24 (147,102)24 (147,152)24 (140,52) g_N (140,100) g_2 (140,150) g_1 (159,102)(4,1)91 (159,152)(3,-

2)96 (159,54)(3,2)96 (159,54)(4,1)92 (159,102)(4,-1)92 (159,152)(4,-1)92 (175,80) w_N (175,115) w_2 (175,149) w_1 (262,78)24 (262,128)24 (255,125) S (255,75) D (274,128)(2,-1)40 (274,78)(2,1)40 (326,102)24 (319,100) h (338,102)(1,0)40 (380,100) y

Figure 2. A GRNN with one output. g represents a Gaussian activation function, S an S-summation neuron, $S=\sum_{n=1}^{N} w_n g(\mathbf{x})$ and D a D-summation neuron, $D=\sum_{n=1}^{N} g(\mathbf{x})$. h carries out the computation $y = S \div D$.

2. Generalized regression neural network

The generalized regression NN (GRNN) is a modification of the standard Gaussian radial basis function (GRBF) NN [20] and is designed specifically to facilitate function approximation for system modelling and prediction. The architecture is shown in figure 2 for a scalar output. The centre, $\mathbf{c}_n, n = 1, 2, \ldots, N$ of each GRBF is an input vector from the training set, and each weight w_n is the target associated with the respective input vector i.e. $w_n = t_n$. The output, y of the GRNN is given by

$$y = \frac{\sum_{n=1}^{N} w_n g(\mathbf{x})}{\sum_{n=1}^{N} g(\mathbf{x})} \tag{6}$$

where

$$g(\mathbf{x}) = \exp(-\| \mathbf{c}_1 - \mathbf{x}_n \| / \sigma^2).$$

The value assigned to σ controls the generalization capability of the RBF NN. As a rule of thumb, σ should be greater than the minimum Euclidean distance between pairs of data and smaller than the maximum distance between pairs. A value commonly used is

$$\sigma = \frac{d_{max}}{\sqrt{C}} \tag{7}$$

where d_{max} is the maximum Euclidean distance between input vectors and C is the number of GRBFs chosen for the network. [20] This value can produce a good function approximation where data points are evenly distributed throughout the input space. Clearly, the magnitude of the GRNN output is bounded by the size of the maximum value of $w_n \equiv t_n$ i.e.

$$| y | \le \arg \max_{n} | w_n | . \tag{8}$$

This contrasts with MLP training, where there is no guaranteed *a priori* bound on outputs due to the nature of the learning process.

3. Generalization

Care must be taken to ensure that good generalization is built into a NN, so that when presented with a previously unseen input it will produce an appropriate output. In practice, unless virtual data [23] is used for training, the data will contain noise. The degree of noise affects the ability of a NN to model the true underlying input-output mapping. A NN trained to a very small error may have effectively ‘rote-memorized’ training data (including noise) without learning the true function. To prevent this, it is sometimes better to interrupt training at some point before the pre-determined MSE target is reached. The best statistical approach is to split data into a training subset and a validation subset. [20] The first subset is used to train the NN, whilst the second is used to test MSE $_{val}$, the validation error defined by

$$\mathrm{MSE}_{val} = \sum_{i=1}^{N} (t_i^{val} - y_i)^2) \tag{9}$$

of the temporarily ‘frozen’ network at intervals in the training procedure, where t^{val} is a target from the validation set. This is known as cross-validation. If a point is reached where the error of the validation set begins to rise (typically, a few epochs of consecutive increasing error is necessary if MSE_{val} is not monotonically decreasing), the network training is stopped. In the context of an adaptive TSP MLP, the use of on-line validation data may be impractical. There are 3 options to consider:

1. Train using the most recent data, then validate using older data
2. Train using older data, then validate using the most recent data
3. Train using a random selection of recent data points, then validate using the data unused for training

Clearly the first two methods may give poor results for rapidly changing non-stationary signals, since there is no guarantee that the validation set is within the domain of the training data. The third method can sometimes produce an ‘average’ network which may miss a rapid change occurring in the most recent data samples. This is, however, the only practical method for implementing on-line where a signal is non-stationary, and is effective where data is noisy. In the present work, cross-validation is implemented by keeping 1 in 5 data points for validation purposes, whilst training the NN on the remaining 4 out of 5 points (see figure 3). This ratio is considered optimal for most NN training applications.[24] The validation points are sampled at uniform intervals to avoid the possibility of bias.

[scale=0.6]NN $_s ampling1.eps$

Figure 3. **Demonstration of data sampling for validation and RBF centres.**

4. Training algorithms

There are a large number of algorithms available to train NNs. A good introduction to several of these is provided by [25]. For adaptive updating of a TSP NN, a fast algorithm is required, since any delay limits the ability of the NN to provide accurate prediction (cf. Kakar *et al*[15] where training takes $\square 120$ s on 20 s of data for 0.24 s ahead prediction). A brief description is given below of three training algorithms which have become popular due to their rapid convergence characteristics. The first two described have been used previously within the context of tumour motion prediction.[12,14]

1. Conjugate-gradient backpropagation algorithm

The conjugate-gradient (CG) search method has been shown to guarantee convergence to the minimum of a quadratic function in a finite number of iterations $\leq W$ where W is the number of parameters of the function.[26] The energy function, F, is not quadratic in the NN weights, so there is no guaranteed convergence of CG to the global error minimum. However, a modification of CG called conjugate-gradient backpropagation (CGBP) can be used to train a MLP. CGBP employs a line search to find successive local minimum values of F. At each local minimum, the algorithm is re-set by calculating a new search direction until the algorithm either converges or is terminated by a suitable criterion e.g. time limitation or validation error. Sharp *et al*[12] use CGBP to train a static MLP.

2. Levenberg-Marquardt

The Levenberg-Marquardt (LM) algorithm[27,28] is designed specifically to minimize the sum-of-squares error function, using a formula that (partly) assumes that the underlying function modelled by the network is linear. Close to a minimum this assumption is approximately true, and the algorithm can make very rapid progress. Further away it may be a very poor assumption. LM therefore compromises between the linear model and a gradient-descent approach. A move is only accepted if it improves the error, and if necessary the gradient-descent model[29] is used with a sufficiently small step to guarantee downhill movement. The LM algorithm requires the inversion of $[\mathbf{J}_k^T\mathbf{J}_k + \lambda_k\mathbf{I}]$ at the k^{th} iteration, where **J** is the Jacobian matrix and λ is an adjustable scalar parameter. The size of this matrix is dependent on the number of training pairs, N, presented to the NN and the computational cost of the required matrix inversion scales as $O(N^3)$.[30] Calculation time for an epoch using LM may therefore be considerably longer than that for an epoch using CGBP. For the present work, at the k^{th} iteration, the value of λ is updated according to the rule

$$\begin{cases} \lambda_{k+1} = 0.1\lambda_k, F_{k+1} < F_k \\ \lambda_{k+1} = 10\lambda_k, F_{k+1} \geq F_k \end{cases} \tag{10}$$

LM is used by Isaksson *et al*[14] for on-line training of a MLP for IGRT.

3. GRNN training

The method for constructing the GRNN is described above. It should be noted that iterating is not required, so that a GRNN can be produced in considerably less time than required to produce a comparable TSP MLP.

5. TSP MLP parameterization and training

In general, one hidden layer of neurons is sufficient for all but the most complex of input/output relationships,[31] so is implemented for all architectures. Tansig neurons are used for the hidden layer and linear neurons on the output layer. Because of the large number of combinations of inputs and hidden neurons possible, an average is made using the first 10 s of data from a random sample of 30 breathing traces. Since the data is sampled at 30 Hz, this translates to $10 \times 30 = 300$ data points. Subsequently, training is implemented for each of the breathing traces using CGBP, LM and BR using the following protocol:

1. Train the TSP MLP using the first 10 s of data. CGBP and LM use 5 s of data for training and 5 s for validation. BR uses all 10 s for training.
2. Subsequently up-date at 1 s intervals for 0.2 s only, using the latest 5 s of data.

Training is first carried out using unfiltered data, then filtered data. In the present work validation is only used for initial training using the CGBP and LM algorithms off-line. On-line adaptation is carried out using a limited number of epochs for each algorithm, in order to minimize the possibility of over-fitting.

6. Error evaluation

The root-mean-squared error (RMSE) between predicted and observed marker position is a standard metric used in IGRT.[12,13,15] It is defined for N observations by

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i^{pred} - \mathbf{x}_i^{ob})^2} \tag{11}$$

where $\mathbf{x}^{pred}$ is predicted, and $\mathbf{x}^{ob}$ observed, marker position. N.B. RMSE $\equiv$ standard deviation of the error vector $\mathbf{e} = [\mathbf{x}^{pred} - \mathbf{x}^{ob}]$. When using unfiltered data for training, RMSE is calculated using the observed i.e. unfiltered observation so that the value is not artificially reduced. The average RMSE for all 331 breathing traces

$$\overline{\text{RMSE}} = \frac{1}{331}\sum_{j=1}^{331}(\text{RMSE})_j \tag{12}$$

is then used as the final RMSE measure. Error is also presented in terms of the magnitude of the maximum single error over all breathing traces, given by

$$|e^{max}|^* = \arg\max_j |e_j^{max}|, j = 1, 2, \ldots, 331 \tag{13}$$

and the average value of the maximum errors

$$\overline{|e^{max}|} = \frac{1}{331}\sum_{j=1}^{331}|e_j^{max}| \tag{14}$$

All training is carried out using the Matlab ® 6.5 Neural Network toolbox version 4.0.2.

3. Results and discussion

Table 1. Optimization results for TSP MLPs - unfiltered data

.2 s latency					
Training algorithm	CG	LM	BR+CG	GRNN	No prediction
$\overline{\mathrm{RMSE}}$ (mm)	0.727	1.608	0.765	0.780	1.196
Standard deviation of RMSE (mm)	0.547	10.857	0.715	0.166	0.326
$\lvert e^{max} \rvert^*$ (mm)	15.91	321.6	15.80	8.339	6.256
$\overline{\lvert e^{max} \rvert}$ (mm)	3.283	84.26	4.279	5.353	3.676
No. of inputs	4	3	3	2	—
σ	–	-	–	0.031	–
No. of hidden neurons	8	7	8	30	—
.4 s latency					
Training algorithm	CG	LM	BR+CG	GRNN	No prediction
$\overline{\mathrm{RMSE}}$ (mm)	1.092	2.267	1.341	1.696	2.276
Standard deviation of RMSE (mm)	0.700	23.87	0.879	0.505	0.652
$\lvert e^{max} \rvert^*$ (mm)	18.58	1750	20.78	15.83	10.90
$\overline{\lvert e^{max} \rvert}$ (mm)	4.241	158.8	6.852	4.377	5.750
No. of inputs	4	4	3	2	—
σ	–	-	–	0.031	–
No. of hidden neurons	10	10	9	30	—

A table of results for non-filtered data is given in table 1 with table 2 presenting the results using filtered data. The inferior results for the LM algorithm are attributable to much higher maximum errors in all four categories. This gives an indication of the unsuitability of LM as a robust algorithm for on-line updating. Based on the RMSE criterion, both the CGBP and BR+CG algorithms are significantly superior to using either the LM algorithm or no prediction ($p < 0.006$). Best RMSE results for 0.2 s and 0.4 s latency are obtained using BR ($p < 0.0071$) with filtered input. This indicates that the raw data is noisy.

Table 2. Optimization results for TSP MLPs - filtered data

.2 s latency

Training algorithm	CG	LM	BR+CG	GRNN	No prediction
$\overline{\mathrm{RMSE}}$ (mm)	0.626	0.755	0.484	0.773	1.182
Standard deviation of RMSE (mm)	0.538	5.229	0.542	0.164	0.327
$\lvert e^{max} \rvert^{*}$ (mm)	15.62	228.3	14.81	8.297	5.584
$\overline{\lvert e^{max} \rvert}$ (mm)	3.178	40.82	3.082	5.253	3.152
No. of inputs	4	3	3	2	—
σ	–	-	–	0.023	–
No. of hidden neurons	8	7	8	30	—

.4 s latency

Training algorithm	CG	LM	BR+CG	GRNN	No prediction
$\overline{\mathrm{RMSE}}$ (mm)	1.202	1.614	0.970	1.688	2.264
Standard deviation of RMSE (mm)	0.931	8.402	0.783	0.502	0.652
$\lvert e^{max} \rvert^{*}$ (mm)	18.40	340.6	19.26	15.83	10.45
$\overline{\lvert e^{max} \rvert}$ (mm)	5.027	66.56	5.661	4.346	5.310
No. of inputs	4	4	3	2	—
σ	–	-	–	0.023	–
No. of hidden neurons	10	10	9	30	—

Analysis of maximum error results shows that using no prediction minimizes the worst prediction errors. The GRNN consistently produces best maximum absolute error results amongst the NN prediction methods. From a quality assurance standpoint, the GRNN also has an advantage over the MLP methods in having a bounded output. The radiobiological effect caused by any tracking errors is largely patient specific, which may determine which of the prediction methods, if any, should be used. It is clear, however, that use of the LM algorithm as the sole method for training and updating cannot be justified for this specific application. An investigation of the maximum magnitude of layer weights and biases associated with each algorithm demonstrate the very much higher magnitude of weights induced by training using LM. This explains the disproportionately high errors which occur as a result of relatively small changes in breathing pattern.

Table 3. Analysis of weights and biases - filtered data

.2 s latency

Training algorithm	CG	LM	BR+CG	GRNN
$\lvert weight^{max} \rvert^{*}$	1.884	28.71	9.316	12.33
$\overline{\lvert weight^{max} \rvert}$	1.253	7.346	4.372	7.03
Standard	0.274	6.528	2.173	3.280

deviation of $\lvert weight^{max} \rvert$		
$\lvert bias^{max} \rvert^*$	66.38354.811.81	-
$\overline{\lvert bias^{max} \rvert}$	30.7237.582.848	-
Standard deviation of $\lvert bias^{max} \rvert$	20.0362.482.536	-
.4 s latency		
Training algorithm	CG LM BR+CG	GRNN
$\lvert weight^{max} \rvert^*$	2.23844.4110.94	12.33
$\overline{\lvert weight^{max} \rvert}$	1.21310.924.638	7.030
Standard deviation of $\lvert weight^{max} \rvert$	0.3579.8262.657	3.280
$\lvert bias^{max} \rvert^*$	93.28216.728.00	-
$\overline{\lvert bias^{max} \rvert}$	28.6741.975.855	-
Standard deviation of $\lvert bias^{max} \rvert$	20.8346.526.418	-

4. Conclusions

A study has been made of four different algorithms used to train TSP NNs for tracking tumour movement by external marker. It has been found that assessing the performance of an algorithm soley based on RMSE can potentially lead to unsafe conclusions in terms of the relative advantage of the NN approach. New error criteria have therefore been devised to highlight error maxima. A hybrid algorithm combining BR and CGBP produces best average RMSE results. By minimizing the magnitude of network weights in addition to minimizing output error, average tracking error is reduced significantly compared with that achieved by NN training algorithms previously used in IGRT. Further reduction in RMSE is achieved by implementing an average filter on inputs. A future comparison of filters would be advantageous in order to find an optimal filter to minimize error for this application. Isolated relatively high error values still occur when using the BR+CG algorithm, but with an order of magnitude lower than those observed when using the LM algorithm. For that reason it may be preferable to implement the GRNN, since the maximum error can then at least be stated in advance. An analysis of the magnitude of NN layer weights produced using each algorithm provides some insight into why the LM algorithm maximum error results are higher than those produced by the other algorithms.
0in0in

1. References

1. [1]M. van Herk, “Errors and Margins in Radiotherapy,” Seminars in Radiation Oncology, , 52-64 (2004).

2. [2] D.J. Kim, B.R. Murray, R. Halperin, W.H. Roa, “Held-breath self-gating technique for radiotherapy of non- small-cell lung cancer: a feasibility study,” Int. J. Rad. Oncol. Biol. Phys. , 43-49 (2001).

3. [3]D. Yan, F. Vicini, J. Wong, Martinez, “Adaptive radiation therapy,” Phys. Med. Biol. , 123-132 (1997).

4. [4] D.A. Jaffray, “Emergent Technologies for 3-Dimensional Image-Guided Radiation Delivery,” *Seminars in Radiation Oncology* (Elsevier, 2005).

5. [5]M. Fuss, B.J. Salter, S.X. Cavanaugh, C. Fuss, A. Sadeghi, C.D. Fuller, A. Amerduri, J.M. Hevezi, T.S. Herman, C.R. Thomas Jnr, “Daily ultrasound-based image-guided targeting for radiotherapy of upper abdominal malignancies,” Int. J. of Radiat. Oncol.

Biol. Phys. , 1245-56 (2004).

6. [6] E. Matsinos, M. Endo, R. Kohno, S. Minohara, K. Kohno, H. Asakura, H. Fujiwara, K. Murase, “Current status of the CBCT project at Varian Medical Systems,” Progress in Biomedical Optics and Imaging - Proc. of SPIE , Medical Imaging 2005 - Physics of Medical Imaging, 340-351 (2005).

7. [7] S. Mori S, “Respiratory-gated segment reconstruction for radiation treatment planning using 256-slice CT-scanner during free breathing,” Progress in Biomedical Optics and Imaging - Proc. of SPIE , Medical Imaging 2005 - Physics of Medical Imaging, 711-721 (2005).

8. [8] H. Shirato, Y. Seppenwoolde, K. Kitamura, R. Onimura, S. Shimizu, “Intrafractional Tumor Motion: Lung and Liver,” Seminars in Radiation Oncology , 10-18 (2004).

9. [9] R.I. Berbeco, S. Nishioka, H. Shirato, G.T.Y. Chen, S.B. Jiang, “Residual motion of lung tumours in gated radiotherapy with external respiratory surrogates,” Phys. Med. Biol. , 3655-3667 (2005).

10. [10] L. Dietrich, T. Tücking, S. Nill, U. Oelfke, “Compensation for respiratory motion by gated radiotherapy: an experimental study,” *Phys. Med. Biol.* , 2405-2414 (2005).

11. [11] A.E. Lujan, E.W. Larsen, J.M. Balter, R.K. Ten Haken, “A method for incorporating organ motion due to breathing into 3D dose calculations,” *Med. Phys.* , 715-720 (1999).

12. [12] G.C. Sharp, S.B. Jiang, S. Shimizu, H. Shirato, “Prediction of respiratory tumour motion for real-time image-guided radiotherapy,” *Phys. Med. Biol.* , 425-440 (2004).

13. [13] S. Vedam, A. Docef, D.A. Todor, V.R. Kini, R. Mohan, “Predicting respiratory motion for four-dimensional radiotherapy,” Med. Phys. 2274-2283 (2004).

14. [14] M. Isaksson, J. Jalden, M.J. Murphy, “On using an adaptive neural network to predict lung tumor motion during respiration for radiotherapy applications,” Med. Phys. , 3801-3809 (2005).

15. [15] M. Kakar, H. Nystrom, L.R. Aarup, T.J. Nøttrup, D.R. Olsen, “Respiratory motion prediction by using the adaptive neuro fuzzy inference system (ANFIS),” Phys. Med. Biol. , 4721-4728 (2005).

16. [16] H. Yan, F.-F. Yin, G.-P. Zhu, M. Ajlouni, J.H. Kim, “Adaptive prediction of internal target motion using external marker motion: a technical study,” Phys. Med. Biol. , 31-44 (2006).

17. [17] L.H. Tsoukalas, R.E. Uhrig, *Fuzzy and Neural Approaches in Engineering* (New York: Wiley, 1997).

18. [18] R. George, S.S. Vedam, T.D. Chung, V. Ramakrishnan, P.J. Keall “The application of the sinusoidal model to lung cancer patient respiratory motion,” *Med. Phys.* , 2850-2861 (2005).

19. [19] J.W. Hines, *MATLAB Supplement to Fuzzy and Neural Approaches in Engineering*, (New York: Wiley, 1997).

20. [20] S. Haykin *Neural Networks: A Comprehensive Foundation,* 2nd *ed.* (Prentice-Hall Inc, 1999).

21. [21] G.P. Liu *Nonlinear Identification and Control* (Springer-Verlag London, Advances in Industrial Control Monograph, 2001).

22. [22] K.G. Beauchamp, C.K. Yuen, *Digital methods for signal analysis* (George Allen & Unwin Ltd, 1979).

23. [23] J.H. Goodband, O.C.L. Haas, J.A. Mills, “A mixture of experts committee machine to design compensators for intensity modulated radiation therapy,” Pattern Recognition , 1704-1714 (2006).

24. [24] M. Kearns “A bound on the error of cross-validation using the approximation and estimation rates, with consequences for the training-test split,” Advances in Neural Information Processing Systems , 183-189 (1996).

25. [25] M.T. Hagen, H.B. Demuth, M. Beale, *Neural Network Design* (PWS Publishing Company, 1996)

26. [26] P.E. Gill, W. Murray, M.H. Wright, *Practical Optimization* (New York: Academic Press, 1981).

27. [27] K. Levenberg, “A Method for the solution of certain problems in least squares,” Quarterly Applied Mathematics 164-168 (1944).

28. [28] D. Marquardt, 1963 “An algorithm for least-squares estimation of nonlinear parameters,” SIAM J.(of Applied Mathematics 431-441 (1963).

29. [29] D.E. Rumelhardt, J.L. McClelland, *Parallel Distributed Processing* (MIT Press, Cambridge, Mass, 1986).

30. [30] V. Tresp, Committee Machines, *Handbook for Neural Network Signal Processing*, ed. Hu Y H, Hwang J-N (CRC Press, 2001).

31. [31] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, 1991)

32. MacKay D 1992 A practical Bayesian framework for back-propagation networks, *Neural Computation* 448-472